

Student projects for *Murach's* *Python Programming*

These projects let you practice the skills that you learn as you progress through *Murach's Python Programming*. These projects provide a range of difficulty levels, and your instructor will assign selected projects as you progress through this course.

In the project names, the first number specifies the chapter that you should complete before starting the exercise. For example, you should complete chapter 3 before starting project 3-1 or 3-2, and you should complete chapter 7 before starting project 7-1, 7-2, or 7-3.

General guidelines	3
Project 2-1: Student Registration	4
Project 2-2: Pay Check Calculator	5
Project 2-3: Tip Calculator	6
Project 2-4: Price Comparison	7
Project 2-5: Travel Time Calculator	8
Project 3-1: Letter Grade Converter	9
Project 3-2: Tip Calculator	10
Project 3-3: Change Calculator	11
Project 3-4: Shipping Calculator	12
Project 3-5: Table of Powers	13
Project 4-1: Even or Odd Checker	14
Project 4-2: Hike Calculator	15
Project 4-3: Feet and Meters Converter	16
Project 4-4: Sales Tax Calculator	17
Project 4-5: Dice Roller	18
Project 4-6: Prime Number Checker	19
Project 5-1: Tax Calculator (Debug)	20
Project 5-2: Guessing Game (Debug)	21
Project 6-1: Prime Number Checker	21
Project 6-2: Wizard Inventory	23
Project 6-3: Contact Manager	24
Project 6-4: Quarterly Sales	25
Project 6-5: Tic Tac Toe	26
Project 7-1: Pig Game Rules	27
Project 7-2: Wizard Inventory	28
Project 7-3: Contact Manager	29
Project 7-4: Monthly Sales	30
Project 7-5: Email List Cleaner	31
Project 8-1: Tip Calculator	32
Project 8-2: Wizard Inventory	33
Project 8-3: Contact Manager	34
Project 8-4: Monthly Sales	35

Project 9-1: Interest Calculator	36
Project 9-2: Monthly Payment Calculator.....	37
Project 9-3: Sales Report.....	38
Project 9-4: Aircraft Fuel Calculator	39
Project 10-1: HTML Converter	40
Project 10-2: Email Creator.....	41
Project 10-3: Interest Calculator	42
Project 10-4: Pig Latin Translator	43
Project 11-1: Birthday Calculator	44
Project 11-2: Arrival Time Estimator	45
Project 11-3: Future Value Logger	46
Project 12-1: Game Stats.....	47
Project 12-2: Bird Counter	48
Project 12-3: Champion Counter	49
Project 12-4: Monthly Sales	50
Project 13-1: Greatest Common Divisor	51
Project 13-2: Tree Pattern.....	52
Project 13-3: Binary Search	53
Project 13-4: Number Finder.....	54
 Project 14-1: Rectangle Calculator	 54
Project 14-2: Card Dealer	56
Project 14-3: Customer Viewer	57
Project 15-1: Rectangle or Square Calculator.....	58
Project 15-2: Roshambo	59
Project 15-3: Customer or Employee Creator	60
Project 15-4: Random Integer List	61
Project 16-1: Blackjack	62
 Project 17-1: Customer Data Importer	 63
Project 17-2: Player Manager	64
Project 17-3: Task List	65
Project 17-4: Product Manager	66
Project 18-1: Right Triangle Calculator	67
Project 18-2: Preferences	68

General guidelines

Naming

- When creating the filenames for your programs, please use the convention specified by your instructor. Otherwise, for programs that consist of just one file, name the file *first_last_program.py* where *first_last* specifies your first and last name and *program* specifies the name of the program. For programs that have multiple files, store the files in a folder named *first_last_program*.
- When creating names for variables and functions, please use the guidelines and recommendations specified by your instructor. Otherwise, use the guidelines and recommendations specified in *Murach's Python Programming*.

User interfaces

- You should think of the user interfaces that are shown for the projects as starting points. If you can improve on them, especially to make them more user-friendly, by all means do so.

Specifications

- You should think of the specifications that are given for the projects as starting points. If you have the time to enhance the programs by improving on the starting specifications, by all means do so.

Top-down development

- Always start by developing a working version of the program for a project. That way, you'll have something to show for your efforts if you run out of time. Then, you can build out that starting version of the program until it satisfies all of the specifications.
- From chapter 5 on, you should use top-down coding and testing as you develop your programs. You might also want to sketch out a hierarchy chart for each program as a guide to your top-down coding.

Files supplied by your instructor

- Some of the projects require starting text, CSV, or binary files. These files are identified in the specifications for the projects, and your instructor should make these starting files available to you.

Project 2-1: Student Registration

Create a program that allows a student to complete a registration form and displays a completion message that includes the user's full name and a temporary password.

Console

```
Registration Form

First name:      Eric
Last name:       Idle
Birth year:      1934

Welcome Eric Idle!
Your registration is complete.
Your temporary password is: Eric*1934
```

Specifications

- The user's full name consists of the user's first name, a space, and the user's last name.
- The temporary password consists of the user's first name, an asterisk (*), and the user's birth year.
- Assume the user will enter valid data.

Project 2-2: Pay Check Calculator

Create a program that calculates a user's weekly gross and take-home pay.

Console

```
Pay Check Calculator

Hours Worked:    35
Hourly Pay Rate: 14.50

Gross Pay:       507.5
Tax Rate:        18%
Tax Amount:      91.35
Take Home Pay:   416.15
```

Specifications

- The formula for calculating gross pay is:
`gross pay = hours worked * hourly rate`
- The formula for calculating tax amount is:
`tax amount = gross pay * (tax rate / 100)`
- The formula for calculating take home pay is:
`take home pay = gross pay - tax amount`
- The tax rate should be 18%, but the program should store the tax rate in a variable so that you can easily change the tax rate later, just by changing the value that's stored in the variable.
- The program should accept decimal entries like 35.5 and 14.25.
- Assume the user will enter valid data.
- The program should round the results to a maximum of two decimal places.

Project 2-3: Tip Calculator

Create a program that calculates the tip and total for a meal at a restaurant.

Console

```
Tip Calculator

Cost of meal: 52.31
Tip percent: 20

Tip amount: 10.46
Total amount: 62.77
```

Specifications

- The formula for calculating the tip amount is:
`tip = cost of meal * (tip percent / 100)`
- The program should accept decimal entries like 52.31 and 15.5.
- Assume the user will enter valid data.
- The program should round the results to a maximum of two decimal places.

Project 2-4: Price Comparison

Create a program that compares the unit prices for two sizes of laundry detergent sold at a grocery store.

Console

```
Price Comparison

Price of 64 oz size: 5.99
Price of 32 oz size: 3.50

Price per oz (64 oz): 0.09
Price per oz (32 oz): 0.11
```

Specifications

- The formula for calculating price per ounce is:
`price per ounce = price / ounces`
- Assume the user will enter valid data.
- The program should round the results to a maximum of two decimal places.

Project 2-5: Travel Time Calculator

Create a program that calculates the estimated hours and minutes for a trip.

Console

```
Travel Time Calculator

Enter miles: 200
Enter miles per hour: 65

Estimated travel time
Hours: 3
Minutes: 5
```

Specifications

- The program should only accept integer entries like 200 and 65.
- Assume that the user will enter valid data.

Hint

- Use integers with the integer division and modulus operators to get hours and minutes.

Project 3-1: Letter Grade Converter

Create a program that converts number grades to letter grades.

Console

```
Letter Grade Converter

Enter numerical grade: 90
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 88
Letter grade: A

Continue? (y/n): y

Enter numerical grade: 80
Letter grade: B

Continue? (y/n): y

Enter numerical grade: 67
Letter grade: C

Continue? (y/n): y

Enter numerical grade: 59
Letter grade: F

Continue? (y/n): n

Bye!
```

Specifications

- The grading criteria is as follows:

A	88-100
B	80-87
C	67-79
D	60-66
F	<60
- Assume that the user will enter valid integers for the grades.
- The program should continue only if the user enters “y” or “Y” to continue.

Project 3-2: Tip Calculator

Create a program that calculates three options for an appropriate tip to leave after a meal at a restaurant.

Console

```
Tip Calculator

Cost of meal: 52.31

15%
Tip amount:    7.85
Total amount:  60.16

20%
Tip amount:    10.46
Total amount:  62.77

25%
Tip amount:    13.08
Total amount:  65.39
```

Specifications

- The program should calculate and display the cost of tipping at 15%, 20%, or 25%.
- Assume the user will enter valid data.
- The program should round results to a maximum of two decimal places.

Project 3-3: Change Calculator

Create a program that calculates the coins needed to make change for the specified number of cents.

Console

```
Change Calculator

Enter number of cents (0-99): 99

Quarters: 3
Dimes:    2
Nickels:  0
Pennies:  4

Continue? (y/n): y

Enter number of cents (0-99): 55

Quarters: 2
Dimes:    0
Nickels:  1
Pennies:  0

Continue? (y/n): n

Bye!
```

Specifications

- The program should display the minimum number of quarters, dimes, nickels, and pennies that one needs to make up the specified number of cents.
- Assume that the user will enter a valid integer for the number of cents.
- The program should continue only if the user enters “y” or “Y” to continue.

Project 3-4: Shipping Calculator

Create a program that calculates the total cost of an order including shipping.

Console

```
=====
Shipping Calculator
=====
Cost of items ordered: 49.99
Shipping cost:         7.95
Total cost:            57.94

Continue? (y/n): y
=====
Cost of items ordered: -65.50
You must enter a positive number. Please try again.
Cost of items ordered: 65.50
Shipping cost:         9.95
Total cost:            75.45

Continue? (y/n): n
=====
Bye!
```

Specifications

- Use the following table to calculate shipping cost:

COST OF ITEMS	SHIPPING COST
=====	
< 30.00	5.95
30.00-49.99	7.95
50.00-74.99	9.95
> 75.00	FREE

- If the user enters a number that's less than zero, display an error message and give the user a chance to enter the number again.

Project 3-5: Table of Powers

Create a program that displays a table of squares and cubes for the specified range of numbers.

Console

Table of Powers		
Start number: 90		
Stop number: 100		
Number	Squared	Cubed
=====	=====	=====
90	8100	729000
91	8281	753571
92	8464	778688
93	8649	804357
94	8836	830584
95	9025	857375
96	9216	884736
97	9409	912673
98	9604	941192
99	9801	970299
100	10000	1000000

Specifications

- The formulas for calculating squares and cubes are:
`square = x ** 2`
`cube = x ** 3`
- Use tabs to align the columns.
- Assume that the user will enter valid integers.
- Make sure the user enters a start integer that's less than the stop integer. If the user enters a start integer that's greater than the stop integer, display an error message and give the user a chance to enter the integers again.

Project 4-1: Even or Odd Checker

Create a program that checks whether a number is even or odd.

Console

```
Even or Odd Checker  
  
Enter an integer: 33  
This is an odd number.
```

Specifications

- Store the code that gets user input and displays output in the main function.
- Store the code that checks whether the number is even or odd in a separate function.
- Assume that the user will enter a valid integer.

Project 4-2: Hike Calculator

Create a program that converts the number of miles that you walked on a hike to the number of feet that you walked.

Console

```
Hike Calculator
```

```
How many miles did you walk?: 4.5  
You walked 23760 feet.
```

Specifications

- The program should accept a float value for the number of miles.
- Store the code that gets user input and displays output in the main function.
- There are 5280 feet in a mile.
- Store the code that converts miles to feet in a separate function. This function should return an int value for the number of feet.
- Assume that the user will enter a valid number of miles.

Project 4-3: Feet and Meters Converter

Create a program that converts feet to meters and vice versa.

Console

```
Feet and Meters Converter

Conversions Menu:
a. Feet to Meters
b. Meters to Feet
Select a conversion (a/b): a

Enter feet: 100
30.48 meters

Would you like to perform another conversion? (y/n): y

Conversions Menu:
a. Feet to Meters
b. Meters to Feet
Select a conversion (a/b): b

Enter meters: 100
328.08 feet

Would you like to perform another conversion? (y/n): n

Thanks, bye!
```

Specifications

- The formula for converting feet to meters is:
`feet = meters / 0.3048`
- The formula for converting meters to feet is:
`meters = feet * 0.3048`
- Store the code that performs the conversions in functions within a module. For example, store the code that converts feet to meters in a function in a module.
- Store the code that displays the title in its own function, and store the code that displays the menu in its own function, but store the rest of the code that gets input and displays output in a main function.
- Assume the user will enter valid data.
- The program should round results to a maximum of two decimal places.

Project 4-4: Sales Tax Calculator

Create a program that uses a separate module to calculate sales tax and total after tax.

Console

```
Sales Tax Calculator

ENTER ITEMS (ENTER 0 TO END)
Cost of item: 35.99
Cost of item: 27.50
Cost of item: 19.59
Cost of item: 0
Total:           83.08
Sales tax:       4.98
Total after tax: 88.06

Again? (y/n): y

ENTER ITEMS (ENTER 0 TO END)
Cost of item: 152.50
Cost of item: 59.80
Cost of item: 0
Total:           212.3
Sales tax:       12.74
Total after tax: 225.04

Again? (y/n): n

Thanks, bye!
```

Specifications

- The sales tax rate should be 6% of the total.
- Store the sales tax rate in a module. This module should also contain functions that calculate the sales tax and the total after tax. These functions should round the results to a maximum of two decimal places.
- Store the code that gets input and displays output in another module. Divide this code into functions wherever you think it would make that code easier to read and maintain.
- Assume the user will enter valid data.

Project 4-5: Dice Roller

Create a program that uses a function to simulate the roll of a die.

Console

```
Dice Roller

Roll the dice? (y/n): y

Die 1: 3
Die 2: 6
Total: 9

Roll again? (y/n): y

Die 1: 1
Die 2: 1
Total: 2
Snake eyes!

Roll again? (y/n): y

Die 1: 6
Die 2: 6
Total: 12
Boxcars!

Roll again? (y/n): n
```

Specifications

- The program should roll two six-sided dice.
- Store the code that rolls a single die in a function.
- Store the code that gets input and displays output in the main function. Whenever it's helpful, use helper functions to split this code into other functions.
- The program should display a special message for two ones (snake eyes) and two sixes (boxcars).

Project 4-6: Prime Number Checker

Create a program that checks whether a number is a prime number and displays the total number of factors if it is not a prime number.

Console

```
Prime Number Checker

Please enter an integer between 1 and 5000: 1
Invalid integer. Please try again.
Please enter an integer between 1 and 5000: 2
2 is a prime number.

Try again? (y/n): y

Please enter an integer between 1 and 5000: 3
3 is a prime number.

Try again? (y/n): y

Please enter an integer between 1 and 5000: 4
4 is NOT a prime number.
It has 3 factors.

Try again? (y/n): y

Please enter an integer between 1 and 5000: 6
6 is NOT a prime number.
It has 4 factors.

Try again? (y/n): n

Bye!
```

Specifications

- A prime number is only divisible by two factors (1 and itself). For example, 7 is a prime number because it is only divisible by 1 and 7.
- If the number is not a prime number, the program should display its number of factors. For example, 6 has four factors (1, 2, 3, and 6).
- Store the code that gets a valid integer for this program in its own function.
- Store the code that calculates the number of factors for a number in its own function.
- Store the rest of the code that gets input and displays output in the main function.

Project 5-1: Tax Calculator (Debug)

Debug an existing program.

Console

```
Sales Tax Calculator  
  
Total amount: 99.99  
Total after tax: 105.99
```

Specifications

- Your instructor should provide a program file named p5-1_sales_tax.py.
- Your job is to test this program and to find and fix all of the syntax, runtime, and logic errors that it contains.
- The sales tax should be 6% of the total.

Project 5-2: Guessing Game (Debug)

Debug an existing program.

Console

```
Guess the number!

Enter the upper limit for the range of numbers: 100
I'm thinking of a number from 1 to 100.

Your guess: 50
Too low.
Your guess: 75
Too low.
Your guess: 87
Too low.
Your guess: 94
Too low.
Your guess: 97
Too high.
Your guess: 95
Too low.
Your guess: 96
You guessed it in 7 tries.

Play again? (y/n): y

Enter the upper limit for the range of numbers: 10
I'm thinking of a number from 1 to 10.

Your guess: 5
Too low.
Your guess: 7
Too low.
Your guess: 9
Too low.
Your guess: 10
You guessed it in 4 tries.

Play again? (y/n): n

Bye!
```

Specifications

- Your instructor should provide a program file named `p5-2_guesses.py`.
- Your job is to test the program and to find and fix all the errors that it contains.

Project 6-1: Prime Number Checker

Create a program that checks whether a number is a prime number and displays its factors if it is not a prime number.

Console

```
Prime Number Checker

Please enter an integer between 1 and 5000: 5
5 is a prime number.

Try again? (y/n): y

Please enter an integer between 1 and 5000: 6
6 is NOT a prime number.
It has 4 factors: 1 2 3 6

Try again? (y/n): y

Please enter an integer between 1 and 5000: 200
200 is NOT a prime number.
It has 12 factors: 1 2 4 5 8 10 20 25 40 50 100 200

Try again? (y/n): n

Bye!
```

Specifications

- A prime number is divisible by two factors (1 and itself). For example, 7 is a prime number because it is only divisible by 1 and 7.
- If the user enters an integer that's not between 1 and 5000, the program should display an error message.
- If the number is a prime number, the program should display a message.
- If the number is not a prime number, the program should display a message. Then, it should display the number of factors for the number and a list of those factors.
- Store the factors for each number in a list.
- Use functions to organize the code for this program.

Project 6-2: Wizard Inventory

Create a program that keeps track of the items that a wizard can carry.

Console

```
The Wizard Inventory program

COMMAND MENU
show - Show all items
grab - Grab an item
edit - Edit an item
drop - Drop an item
exit - Exit program

Command: show
1. wooden staff
2. wizard hat
3. cloth shoes

Command: grab
Name: potion of invisibility
potion of invisibility was added.

Command: grab
You can't carry any more items. Drop something first.

Command: show
1. wooden staff
2. wizard hat
3. cloth shoes
4. potion of invisibility

Command: edit
Number: 1
Updated name: magic wooden staff
Item number 1 was updated.

Command: drop
Number: 3
cloth shoes was dropped.

Command: exit
Bye!
```

Specifications

- Use a list to store the items. Provide three starting items.
- The wizard can only carry four items at a time.
- For the edit and drop commands, display an error message if the user enters an invalid number for the item.
- When you exit the program, all changes that you made to the inventory are lost.

Project 6-3: Contact Manager

Create a program that a user can use to manage the primary email address and phone number for a contact.

Console

```
Contact Manager

COMMAND MENU
list - Display all contacts
view - View a contact
add - Add a contact
del - Delete a contact
exit - Exit program

Command: list
1. Guido van Rossum
2. Eric Idle

Command: view
Number: 2
Name: Eric Idle
Email: eric@ericidle.com
Phone: +44 20 7946 0958

Command: add
Name: Mike Murach
Email: mike@murach.com
Phone: 559-123-4567
Mike Murach was added.

Command: del
Number: 1
Guido van Rossum was deleted.

Command: list
1. Eric Idle
2. Mike Murach

Command: exit
Bye!
```

Specifications

- Use a list of lists to store the data for the contacts. Provide starting data for two or more contacts.
- For the view and del commands, display an error message if the user enters an invalid contact number.
- When you exit the program, all changes that you made to the contact list are lost.

Project 6-4: Quarterly Sales

Create a program that gets quarterly sales from a user and calculates the total of all four quarters as well as the average, lowest, and highest quarters.

Console

```
The Quarterly Sales program

Enter sales for Q1: 12312.57
Enter sales for Q2: 15293.21
Enter sales for Q3: 14920.95
Enter sales for Q4: 23432.21

Total:                65958.94
Average Quarter:      16489.74
Lowest Quarter:       12312.57
Highest Quarter:      23432.21
```

Specifications

- Use a list to store the sales for each quarter.
- Round the results of each entry to a maximum of 2 decimal digits.

Project 6-5: Tic Tac Toe

Create a two-player Tic Tac Toe game.

Console

```
Welcome to Tic Tac Toe

+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+

X's turn
Pick a row (1, 2, 3): 1
Pick a column (1, 2, 3): 1

+---+---+---+
| X |   |   |
+---+---+---+
|   |   |   |
+---+---+---+
|   |   |   |
+---+---+---+

O's turn
Pick a row (1, 2, 3): 1
Pick a column (1, 2, 3): 2

...

X's turn
Pick a row (1, 2, 3): 3
Pick a column (1, 2, 3): 3

+---+---+---+
| X | O | O |
+---+---+---+
|   | X |   |
+---+---+---+
|   |   | X |
+---+---+---+

X wins!

Game over!
```

Specifications

- Use a list of lists to store the Tic Tac Toe grid.
- If the user picks an invalid row or column or a cell that's already taken, display an error message.
- If there is a winner, the game should display an appropriate message and end. Otherwise, it should continue until the grid is full and end in a tie.

Project 7-1: Pig Dice Rules

Create a program that reads a list of rules from a file and displays them.

Console

```
Pig Dice Rules:  
* See how many turns it takes you to get to 20.  
* Turn ends when player rolls a 1 or chooses to hold.  
* If you roll a 1, you lose all points earned during the turn.  
* If you hold, you save all points earned during the turn.
```

Specifications

- Your instructor should provide a text file named rules.txt.
- Your program should read the text file and display it on the console.

Project 7-2: Wizard Inventory

Create a program that keeps track of the items that a wizard can carry.

Console

```
The Wizard Inventory program

COMMAND MENU
walk - Walk down the path
show - Show all items
drop - Drop an item
exit - Exit program

Command: walk
While walking down a path, you see a scroll of uncursing.
Do you want to grab it? (y/n): y
You picked up a scroll of uncursing.

Command: walk
While walking down a path, you see an unknown potion.
Do you want to grab it? (y/n): y
You can't carry any more items. Drop something first.

Command: show
1. a wooden staff
2. a scroll of invisibility
3. a crossbow
4. a scroll of uncursing

Command: drop
Number: 3
You dropped a crossbow.

Command: exit
Bye!
```

Specifications

- Your instructor should provide a text file named `wizard_all_items.txt` that contains a list of all the items that a wizard can carry.
- When the user selects the walk command, the program should read the items from the file, randomly pick one, and give the user the option to grab it.
- Your program should create another file that stores the items that the wizard is carrying. Make sure to update this file every time the user grabs or drops an item.
- The wizard can only carry four items at a time.
- For the drop command, display an error message if the user enters an invalid number for the item.

Project 7-3: Contact Manager

Create a program that a user can use to manage the primary email address and phone number for a contact.

Console

```
Contact Manager

COMMAND MENU
list - Display all contacts
view - View a contact
add - Add a contact
del - Delete a contact
exit - Exit program

Command: list
1. Guido van Rossum
2. Eric Idle

Command: view
Number: 2
Name: Eric Idle
Email: eric@ericidle.com
Phone: +44 20 7946 0958

Command: add
Name: Mike Murach
Email: mike@murach.com
Phone: 559-123-4567
Mike Murach was added.

Command: list
1. Guido van Rossum
2. Eric Idle
3. Mike Murach

Command: exit
Bye!
```

Specifications

- Your instructor should provide a CSV file named contacts.csv.
- When the program starts, it should read the contacts from the CSV file.
- For the view and del commands, display an error message if the user enters an invalid contact number.
- When you add or delete a contact, the change should be saved to the CSV file immediately. That way, no changes are lost, even if the program crashes later.

Project 7-4: Monthly Sales

Create a program that reads the sales for 12 months from a file and calculates the total yearly sales as well as the average monthly sales. In addition, this program should let the user edit the sales for any month.

Console

```
Monthly Sales program

COMMAND MENU
monthly - View monthly sales
yearly  - View yearly summary
edit    - Edit sales for a month
exit    - Exit program

Command: monthly
Jan - 14317
Feb - 3903
Mar - 1073
Apr - 3463
May - 2429
Jun - 4324
Jul - 9762
Aug - 15578
Sep - 2437
Oct - 6735
Nov - 88
Dec - 2497

Command: yearly
Yearly total:      66606
Monthly average:  5550.5

Command: edit
Three-letter Month: Nov
Sales Amount: 8854
Sales amount for Nov was modified.

Command: exit
Bye!
```

Specifications

- Your instructor should provide a CSV file named `monthly_sales.csv` that contains the month and sales data shown above.
- For the edit command, display an error message if the user doesn't enter a valid three-letter abbreviation for the month.
- When the user edits the sales amount for a month, the data should be saved to the CSV file immediately. That way, no data is lost, even if the program crashes later.
- Round the results of the monthly average to a maximum of 2 decimal digits.

Project 7-5: Email List Cleaner

Create a program that reads a CSV file that contains a list of prospects for an email list, reformats the data, and writes the cleaned list to another file.

Console

```
Welcome to the Email List Cleaner

Source list:  prospects.csv
Cleaned list: prospects_clean.csv

Congratulations! Your list has been cleaned!
```

The prospect.csv file

```
FIRST_NAME, LAST_NAME, EMAIL
james, butler, jbutler@gmail.com
Josephine , Darakjy, josephine_darakjy@darakjy.org
ART, VENERE, ART@VENERE.ORG
...
```

The prospect_clean.csv file

```
James, Butler, jbutler@gmail.com
Josephine, Darakjy, josephine_darakjy@darakjy.org
Art, Venere, art@venere.org
...
```

Specifications

- Your instructor should provide a CSV file named `prospects.csv` that contains a list of prospects.
- Your program should fix the formatting problems and write a file named `prospects_clean.csv`.
- All names should use title case. To convert a string to title case, you can call the `title()` method from the string.
- All email addresses should use lowercase. To convert a string to lowercase, you can call the `lower()` method from the string.
- All extra spaces at the start or end of a string should be removed. To do that, you can call the `strip()` method from the string.

Project 8-1: Tip Calculator

Add exception handling to a Tip Calculator program.

Console

```
Tip Calculator

INPUT
Cost of meal: ten
Must be valid decimal number. Please try again.
Cost of meal: -10
Must be greater than 0. Please try again.
Cost of meal: 52.31
Tip percent: 17.5
Must be valid integer. Please try again.
Tip percent: 20

OUTPUT
Cost of meal: 52.31
Tip percent: 20%
Tip amount: 10.46
Total amount: 62.77
```

Specifications

- The program should accept decimal entries like 52.31 and 15.5 for the cost of the meal.
- The program should accept integer entries like 15, 20, 25 for the tip percent.
- The program should validate both user entries. That way, the user can't crash the program by entering invalid data.
- The program should only accept numbers that are greater than 0.
- The program should round results to a maximum of two decimal places.

Project 8-2: Wizard Inventory

Add exception handling to a program that keeps track of the inventory of items that a wizard can carry. If you've done project 7-2, you can add the exception handling to that program. Otherwise, you can start this program from scratch.

Console if the program can't find the inventory file

```
The Wizard Inventory program

COMMAND MENU
walk - Walk down the path
show - Show all items
drop - Drop an item
exit - Exit program

Could not find inventory file!
Wizard is starting with no inventory.

Command: walk
While walking down a path, you see a crossbow.
Do you want to grab it? (y/n): y
You picked up a crossbow.

Command: show
1. a crossbow

Command: drop
Number: x
Invalid item number.

Command:
```

The error message if the program can't find the items file

```
Could not find items file.
Exiting program. Bye!
```

Specifications

- This program should read the text file named `wizard_all_items.txt` that contains all the items a wizard can carry. Your instructor should provide this file if you don't already have it.
- When the user selects the walk command, the program should randomly pick one of the items that were read from the text file and give the user the option to grab it.
- The current items that the wizard is carrying should be saved in an inventory file. Make sure to update this file every time the user grabs or drops an item.
- The wizard can only carry four items at a time. For the drop command, display an error message if the user enters an invalid integer or an integer that doesn't correspond with an item.
- Handle all exceptions that might occur so that the user can't cause the program to crash. If the all items file is missing, display an appropriate error message and exit the program.
- If the inventory file is missing, display an appropriate error message and continue with an empty inventory for the user. That way, the program will write a new inventory file when the user adds items to the inventory.

Project 8-3: Contact Manager

Add exception handling to a program that manages the primary email address and phone number for a contact. If you've done project 7-3, you can add the exception handling to that program. Otherwise, you can start this program from scratch.

Console if the contacts file is not found

```
Contact Manager

Could not find contacts file!
Starting new contacts file...

COMMAND MENU
list - Display all contacts
view - View a contact
add - Add a contact
del - Delete a contact
exit - Exit program

Command: list
There are no contacts in the list.

Command: add
Name: Mike Murach
Email: mike@murach.com
Phone: 559-123-4567
Mike Murach was added.

Command: list
1. Mike Murach

Command: view
Number: 2
Invalid contact number.

Command: view
Number: x
Invalid integer.

Command: view
Number: 1
Name: Mike Murach
Email: mike@murach.com
Phone: 559-123-4567

Command: exit
Bye!
```

Specifications

- When the program starts, it should read the contacts from a CSV file named `contacts.csv`. Your instructor should provide this file if you don't already have it.
- If the program can't find the CSV file, it should display an appropriate message and create a new CSV file that doesn't contain any contact data.
- For the view and del commands, display an appropriate error message if the user enters an invalid integer or an invalid contact number.
- When you add or delete a contact, the change should be saved to the CSV file immediately. That way, no changes are lost, even if the program crashes later.

Project 8-4: Monthly Sales

Add exception handling to a program that reads the sales for 12 months from a file and calculates the total yearly sales as well as the average monthly sales. If you've done project 7-4, you can add the exception handling to that program. Otherwise, you can start this program from scratch.

Console

```
Monthly Sales program

COMMAND MENU
monthly - View monthly sales
yearly  - View yearly summary
edit    - Edit sales for a month
exit    - Exit program

Command: edit
Three-letter Month: Dec
Sales Amount: TK
Sales amount for Dec was modified.

Command: monthly
Jan - 14317
Feb - 3903
Mar - 1073
Apr - 3463
May - 2429
Jun - 4324
Jul - 9762
Aug - 15578
Sep - 2437
Oct - 6735
Nov - 88
Dec - TK

Command: yearly
Using sales amount of 0 for Dec.
Yearly total:      64109
Monthly average:   5342.42

Command: exit
Bye!
```

Specifications

- When the program starts, it should read the sales data from a CSV file named `monthly_sales.csv`. Your instructor should provide this file if you don't already have it.
- If the program can't find the CSV file when it starts, display an error message and exit the program.
- For the edit command, display an error message if the user doesn't enter a valid three-letter abbreviation for the month.
- When the user edits the sales amount for a month, the data should be saved to the CSV file immediately. That way, no data is lost, even if the program crashes later.
- If the CSV file doesn't contain a valid integer for the sales amount for the month, use a value of 0 to calculate the total sales for the year.
- Round the results of the monthly average to a maximum of 2 decimal digits.

Project 9-1: Interest Calculator

Create a program that calculates the interest on a loan.

Console

```
Interest Calculator

Enter loan amount: 520000
Enter interest rate: 5.375

Loan amount:          $520,000.00
Interest rate:         5.375%
Interest amount:       $27,950.00

Continue? (y/n): y

Enter loan amount: 4944.5
Enter interest rate: 1.3

Loan amount:          $4,944.50
Interest rate:         1.300%
Interest amount:       $64.28

Continue? (y/n): n

Bye!
```

Specifications

- The formula for calculating the interest amount is:
$$\text{loan_amount} * (\text{interest_rate} / 100)$$
- Use the Decimal class to make sure that all calculations are accurate. It should round the interest that's calculated to two decimal places, rounding up if the third decimal place is five or greater.
- The interest rate that's displayed can have up to 3 decimal places.
- Assume that the user will enter valid decimal values for the loan amount and interest rate.

Project 9-2: Monthly Payment Calculator

Create a program that calculates the monthly payments on a loan.

Console

```
Monthly Payment Calculator

DATA ENTRY
Loan amount:          500000
Yearly interest rate: 5.6
Years:                30

FORMATTED RESULTS
Loan amount:          $500,000.00
Yearly interest rate: 5.6%
Number of years:      30
Monthly payment:      $2,870.39

Continue? (y/n): y

DATA ENTRY
Loan amount:          500000
Yearly interest rate: 4.3
Years:                30

FORMATTED RESULTS
Loan amount:          $500,000.00
Yearly interest rate: 4.3%
Number of years:      30
Monthly payment:      $2,474.36

Continue? (y/n): n

Bye!
```

Specifications

- The interest rate should only have 1 decimal place for both the calculation and the formatted results.
- The formula for calculating monthly payment is:
$$\text{monthly_payment} = \text{loan_amount} * \text{monthly_interest_rate} / (1 - 1 / (1 + \text{monthly_interest_rate}) ** \text{months})$$
- Assume that the user will enter valid data.

Project 9-3: Sales Report

Create a program that displays a report of sales by quarter for a company with four sales regions (Region 1, Region 2, Region 3, and Region 4).

Console

```
Sales Report

Region      Q1      Q2      Q3      Q4
1           1,540.00  2,010.00  2,450.00  1,845.00
2           1,130.00  1,168.00  1,847.00  1,491.00
3           1,580.00  2,305.00  2,710.00  1,284.00
4           1,105.00  4,102.00  2,391.00  1,576.00

Sales by region:
Region 1: 7,845.00
Region 2: 5,636.00
Region 3: 7,879.00
Region 4: 9,174.00

Sales by quarter:
Q1: 5,355.00
Q2: 9,585.00
Q3: 9,398.00
Q4: 6,196.00

Total annual sales, all regions: $30,534.00
```

Specifications

- The quarterly sales numbers for each region should be hard-coded at the beginning of the program as a list of lists like this:

```
sales = [[1540.0, 2010.0, 2450.0, 1845.0], # Region 1
          [1130.0, 1168.0, 1847.0, 1491.0], # Region 2
          [1580.0, 2305.0, 2710.0, 1284.0], # Region 3
          [1105.0, 4102.0, 2391.0, 1576.0]] # Region 4
```

Project 9-4: Aircraft Fuel Calculator

Create a program that calculates the amount of time and fuel for a 1980 Cessna 172N to fly a specified distance.

Console

```
Aircraft Fuel Calculator

Distance in nautical miles: 180
Flight time: 1 hour(s) and 30 minute(s)
Required fuel: 16.8 gallons

Continue? (y/n): y

Distance in nautical miles: 121
Flight time: 1 hour(s) and 0 minute(s)
Required fuel: 12.7 gallons

Continue? (y/n): n

Bye!
```

Specifications

- Assume that a 1980 Cessna 172N can fly 120 nautical miles (knots) per hour.
- Assume that a 1980 Cessna 172N burns 8.4 gallons of gas per hour.
- For safety, add a half hour to the flight time when calculating the amount of required fuel.
- Round the amount of required fuel to 1 decimal place. For safety, always round up, never down.
- Assume that the user will enter valid data.

Project 10-1: HTML Converter

Create a program that reads an HTML file and converts it to plain text.

Console

```
HTML Converter

Grocery List
* Eggs
* Milk
* Butter
```

Specifications

- Store the following data in a file named groceries.html:

```
<h1>Grocery List</h1>
<ul>
  <li>Eggs</li>
  <li>Milk</li>
  <li>Butter</li>
</ul>
```
- When the program starts, it should read the contents of the file, remove the HTML tags, remove any spaces to the left of the tags, add asterisks (*) before the list items, and display the content and the HTML tags on the console as shown above.

Project 10-2: Email Creator

Create a program that reads a file and creates a series of emails.

Console

```
Email Creator

=====
To:      jbutler@gmail.com
From:    noreply@deals.com
Subject: Deals!

Hi James,

We've got some great deals for you. Check our website!
=====
To:      josephine_darakjy@darakjy.org
From:    noreply@deals.com
Subject: Deals!

Hi Josephine,

We've got some great deals for you. Check our website!
=====
To:      art@venere.org
From:    noreply@deals.com
Subject: Deals!

Hi Art,

We've got some great deals for you. Check our website!
```

Specifications

- Store a list of email addresses in a file using this format:
james,butler,jbutler@gmail.com
josephine,darakjy,josephine_darakjy@darakjy.org
art,venere,art@venere.org
- Store a template for a mass email in a file like this:

```
To:      {email}
From:    noreply@deals.com
Subject: Deals!

Hi {first_name},

We've got some great deals for you. Check our website!
```
- When the program starts, it should read the email addresses and first names from the file, merge them into the mass email template, and display the results on the console.
- All email addresses should be converted to lowercase.
- All first names should be converted to title case.
- If you add names to the list of email addresses, the program should create more emails.
- If you modify the template, the program should change the content of the email that's created.

Project 10-3: Interest Calculator

Create a program that calculates the interest on a loan. This program should make it easy for the user to enter numbers.

Console

```
Interest Calculator

Enter loan amount:  $100,000
Enter interest rate: %2.275

Loan amount:          $100,000.00
Interest rate:         2.275%
Interest amount:      $2,275.00

Continue? (y/n): y

Enter loan amount:   100K
Enter interest rate: 2.275

Loan amount:          $100,000.00
Interest rate:         2.275%
Interest amount:      $2,275.00

Continue? (y/n): n

Bye!
```

Specifications

- Use the Decimal class to make sure that all calculations are accurate. The program should round the interest that's calculated to two decimal places, rounding up if the third decimal place is five or greater.
- The interest rate that's displayed can have up to 3 decimal places.
- Assume that the user will enter valid decimal values for the loan amount and interest rate with these exceptions:
- If the user enters a dollar sign (\$) at the beginning of the loan amount, remove it from the string before converting the string to a number.
- If the user enters a comma in the loan amount, remove it from the string before converting the string.
- If the user enters a K at the end of the loan amount, remove the K from the end of the string, and multiply the loan amount by 1000. For example, a loan amount of 50K should be converted to a value of 50,000.
- If the user enters a percent sign (%) before or after the interest rate, remove it from the string before converting the string to a number.

Project 10-4: Pig Latin Translator

Create a program that translates English to Pig Latin.

Console

```
Pig Latin Translator

Enter text: Tis but a scratch.
English:   tis but a scratch
Pig Latin: istay utbay away atchscray

Continue? (y/n): y

Enter text: We are the knights who say nee!
English:   we are the knights who say nee
Pig Latin: eway areway ethay ightsknay owhay aysay eenay

Continue? (y/n): n

Bye!
```

Specifications

- Convert the English to lowercase before translating.
- Remove any punctuation characters before translating.
- Assume that words are separated from each other by a single space.
- If the word starts with a vowel, just add *way* to the end of the word.
- If the word starts with a consonant, move all of the consonants that appear before the first vowel to the end of the word, then add *ay* to the end of the word.
- If a word starts with the letter *y*, the *y* should be treated as a consonant. If the *y* appears anywhere else in the word, it should be treated as a vowel.

Note

- There are no official rules for Pig Latin. Most people agree on how words that begin with consonants are translated, but there are many different ways to handle words that begin with vowels.

Project 11-1: Birthday Calculator

Create a program that accepts a name and a birth date and displays the person's birthday, the current day, the person's age, and the number of days until the person's next birthday.

Console

```
Birthday Calculator

Enter name: Joel
Enter birthday (MM/DD/YY): 2/4/68
Birthday: Sunday, February 04, 1968
Today:    Tuesday, November 22, 2016
Joel is 48 years old.
Joel's birthday is in 73 days.

Continue? (y/n): y

Enter name: Django
Enter birthday (MM/DD/YY): 12/1/07
Birthday: Saturday, December 01, 2007
Today:    Tuesday, November 22, 2016
Django is 8 years old.
Django's birthday is in 9 days.

Continue? (y/n): y

Enter name: Mike
Enter birthday (MM/DD/YY): 11/22/86
Birthday: Saturday, November 22, 1986
Today:    Tuesday, November 22, 2016
Mike is 30 years old.
Mike's birthday is today!

Continue? (y/n): n
```

Specifications

- Allow the user to enter a date in the MM/DD/YY format. Adjust the date so that it is correct even if the birth year is later than the current year.
- When you calculate the person's age, don't take leap year into account. If the person is more than 2 years old, display the person's age in years. Otherwise, display the person's age in days.
- When you display the message that indicates the number of days to the person's birthday, you can use the following format for a person with a name of John:

```
today      - John's birthday is today!
tomorrow   - John's birthday is tomorrow!
yesterday  - John's birthday was yesterday!
other days - John's birthday is in X days.
```

Project 11-2: Arrival Time Estimator

Create a program that calculates the estimated duration of a trip in hours and minutes. This should include an estimated date/time of departure and an estimated date/time of arrival.

Console

```
Arrival Time Estimator

Estimated date of departure (YYYY-MM-DD): 2016-11-23
Estimated time of departure (HH:MM AM/PM): 10:30 AM
Enter miles: 200
Enter miles per hour: 65

Estimated travel time
Hours: 3
Minutes: 5
Estimated date of arrival: 2016-11-23
Estimated time of arrival: 01:35 PM

Continue? (y/n): y

Estimated date of departure (YYYY-MM-DD): 2016-11-29
Estimated time of departure (HH:MM AM/PM): 11:15 PM
Enter miles: 500
Enter miles per hour: 80

Estimated travel time
Hours: 6
Minutes: 20
Estimated date of arrival: 2016-11-30
Estimated time of arrival: 05:35 AM

Continue? (y/n): n

Bye!
```

Specifications

- For the date/time of departure and arrival, the program should use the YYYY-MM-DD format for dates and the HH:MM AM/PM format for times.
- For the miles and miles per hour, the program should only accept integer entries like 200 and 65.
- Assume that the user will enter valid data.

Project 11-3: Future Value Logger

Create a program that creates a log file that stores the date and time as well as the data for each calculation of a program.

Console

```
Future Value Calculator

Enter monthly investment:      100
Enter yearly interest rate:    3
Enter number of years:        10
Future value:                  14,009.08

Continue? (y/n): y

Enter monthly investment:      100
Enter yearly interest rate:    3.5
Enter number of years:        10
Future value:                  14,385.09

Continue? (y/n): y

Enter monthly investment:      150
Enter yearly interest rate:    3
Enter number of years:        10
Future value:                  21,013.62

Continue? (y/n): n

Bye!
```

Specifications

- Each time the user calculates a future value, write a log file that includes the date/time and data for each calculation. For example, a log file for the calculations shown above would contain data like this:

```
2016-11-28 12:25:28 - 100.0|3.0|10|14009.08
2016-11-28 12:25:35 - 100.0|3.5|10|14385.09
2016-11-28 14:53:12 - 150.0|3.0|10|21013.62
```
- On the console, use commas to separate thousands for the future values, but don't include these commas in the log file.
- Assume that the user will enter valid data.

Project 12-1: Game Stats

Create a program that allows you to view the statistics for a player of a game.

Console

```
Game Stats program

ALL PLAYERS:
Elizabeth
Joel
Mike

Enter a player name: elizabeth
Wins:    41
Losses:  3
Ties:    22

Continue? (y/n): y

Enter a player name: john
There is no player named John.

Continue? (y/n): y

Enter a player name: joel
Wins:    32
Losses:  14
Ties:    17

Continue? (y/n): y

Enter a player name: mike
Wins:    8
Losses:  19
Ties:    11

Continue? (y/n): n

Bye!
```

Specifications

- The program should use a dictionary of dictionaries to store the stats (wins, losses, and ties) for each player. You can code this dictionary of dictionaries at the beginning of the program using any names and statistics that you want. Make sure to provide stats for at least three players.
- The program should begin by displaying an alphabetical list of the names of the players.
- The program should allow the user to view the stats for the specified player.

Project 12-2: Bird Counter

Create a program for birdwatchers that stores a list of birds along with a count of the number of times each bird has been spotted.

Console

```
Bird Counter program

Enter 'x' to exit

Enter name of bird: red-tailed hawk
Enter name of bird: killdeer
Enter name of bird: snowy plover
Enter name of bird: western gull
Enter name of bird: killdeer
Enter name of bird: western gull
Enter name of bird: x
```

Name	Count
=====	=====
Killdeer	2
Red-Tailed Hawk	1
Snowy Plover	1
Western Gull	2

Specifications

- Use a dictionary to store the list of sighted birds and the count of the number of times each bird was sighted.
- Use the pickle module to read the dictionary from a file when the program starts and to write the dictionary to a file when the program ends. That way, the data that's entered by the user isn't lost.

Project 12-3: Champion Counter

Create a program that reads a text file that contains a list of FIFA World Cup champions and determines the country that has won the most championships.

Console

FIFA World Cup Winners		
Country	Wins	Years
=====	=====	=====
Argentina	2	1978, 1986
Brazil	5	1958, 1962, 1970, 1994, 2002
England	1	1966
France	1	1998
Germany	4	1954, 1974, 1990, 2014
Italy	4	1934, 1938, 1982, 2006
Spain	1	2010
Uruguay	2	1930, 1950

Specifications

- Your instructor should provide a text file named `world_cup_champions.txt` that contains data like this:

```
Year,Country,Coach,Captain
1930,Uruguay,Alberto Suppici,José Nasazzi
1934,Italy,Vittorio Pozzo,Gianpiero Combi
1938,Italy,Vittorio Pozzo,Giuseppe Meazza
...
2002,Brazil,Luiz Felipe Scolari,Cafu
2006,Italy,Marcello Lippi,Fabio Cannavaro
2010,Spain,Vicente del Bosque,Iker Casillas
2014,Germany,Joachim Löw,Philipp Lahm
```
- When the program starts, it should read the text file and use a dictionary to store the required data using the name of each country that has won the World Cup as the key.
- The program should compile the data shown above and display the countries alphabetically.

Project 12-4: Monthly Sales

Create a program that allows you to view and edit the sales amounts for each month of the current year.

Console

```
Monthly Sales program

COMMAND MENU
view  - View sales for specified month
edit  - Edit sales for specified month
totals - View sales summary for year
exit  - Exit program

Command: view
Three-letter Month: jan
Sales amount for Jan is 14,317.00.

Command: edit
Three-letter Month: jan
Sales Amount: 15293
Sales amount for Jan is 15,293.00.

Command: totals
Yearly total:      67,855.00
Monthly average:   5,654.58

Command: view
Three-letter Month: july
Invalid three-letter month.

Command: exit
Bye!
```

Specifications

- Your instructor should provide a text file named `monthly_sales.txt` that consists of rows that contain three-letter abbreviations for the month and the monthly sales.
- The program should read the file and store the sales data for each month in a dictionary with the month abbreviation as the key for each item.
- Whenever the sales data is edited, the program should write the changed data to the text file.

Project 13-1: Greatest Common Divisor

Create a program that finds the greatest common divisor of two numbers.

Console

```
Greatest Common Divisor

Number 1: 15
Number 2: 5
Greatest common divisor: 5

Continue? (y/n): y

Number 1: 15
Number 2: 6
Greatest common divisor: 3

Continue? (y/n): y

Number 1: 15
Number 2: 7
Greatest common divisor: 1

Continue? (y/n): n

Bye!
```

Specifications

- Use the following recursive algorithm to calculate the greatest common divisor (GCD):
 divide *x* by *y* and get the remainder
 if the remainder equals 0, GCD is *y* (end function)
 otherwise, calculate GCD again by dividing *y* by remainder
- If number 1 is less than number 2, the program should display a message that indicates that number 1 must be greater than number 2 and give the user another chance to enter the numbers.
- Assume the user will enter valid data.

Project 13-2: Tree Pattern

Create a program that uses tree recursion to print a pattern like the one shown below.

Console

```
Tree Pattern

Enter the number of branches: 5

1 *****
2 *****
1 *****
3 *****
1 *****
2 *****
1 *****
4 *****
1 *****
2 *****
1 *****
3 *****
1 *****
2 *****
1 *****
5 *****
1 *****
2 *****
1 *****
3 *****
1 *****
2 *****
1 *****
4 *****
1 *****
2 *****
1 *****
3 *****
1 *****
2 *****
1 *****
```

Specifications

- The program can only accept a positive number of branches in the tree. Since the number of branches increases exponentially, this program will take a long time to execute for numbers larger than 12 or so.
- Use the following recursive algorithm to generate the pattern shown above:

```
if number = 0, end function
otherwise,
    start branch for number - 1
    print number and its visual representation
    start branch for number - 1
```
- To get the visual representation for a branch, you can multiply the asterisk (*) by 5. In other words, 1 is 5 asterisks, 2 is 10 asterisks, and so on.

Project 13-3: Binary Search

Create a program that uses a recursive function to check whether a number is in a list of sorted numbers.

Console

```
Binary Search

Enter 'x' to exit

Random numbers: [13, 16, 18, 29, 32, 71, 71, 77, 78, 90]

Enter a number from 1 to 100: 1
1 is NOT in random numbers.

Enter a number from 1 to 100: 32
32 is in random numbers.

Enter a number from 1 to 100: 100
100 is NOT in random numbers.

Enter a number from 1 to 100: x
Bye!
```

Specifications

- The program should begin by generating a sorted list of 10 random numbers from 1 to 100.
- The program should allow the user to enter a number from 1 to 100. Then, it should display whether that number is or isn't in the list of random numbers.
- Use the following recursive algorithm to search the list of random numbers:

```
if start index equals end index, target not in list (end function)
calculate middle index
if number at middle index equals target,
    target is in list (end function)
if target number is less than number at middle index,
    search list again from starting index to middle index
if target number is greater than number at the middle index,
    search list again from middle index to end index
```
- The recursive function should only work on sorted lists of numbers. In other words, the program should sort the list before passing it to the recursive function.

Project 13-4: Number Finder

Create a program that finds a number between 0 and 100 by using a recursive function to guess halfway between the high and low limits for the number.

Console

```
Number Finder

Enter 'x' to exit.

Enter a number between 0 and 100: 25
Guess 1 is 50
Guess 2 is 25
The computer found it in 2 guesses.

Enter a number between 0 and 100: 88
Guess 1 is 50
Guess 2 is 75
Guess 3 is 87
Guess 4 is 93
Guess 5 is 90
Guess 6 is 88
The computer found it in 6 guesses.

Enter a number between 0 and 100: 1
Guess 1 is 50
Guess 2 is 25
Guess 3 is 12
Guess 4 is 6
Guess 5 is 3
Guess 6 is 1
The computer found it in 6 guesses.

Enter a number between 0 and 100: 50
Guess 1 is 50
The computer found it in 1 guess!

Enter a number between 0 and 100: x
Bye!
```

Specifications

- The program should allow the user to enter a number between 0 and 100. Then, it should find that number using a binary technique.
- Use the following recursive algorithm to guess the number:
 calculate guess that's halfway between high limit and low limit
 update guess count
 if number equals guess, guess is correct (base case)
 otherwise, if number is less than guess,
 make another guess with same low limit and new high limit
 otherwise, if number is greater than guess,
 make another guess with new low limit and same high limit
- Display a special message if the user gets it in one guess.

Project 14-1: Rectangle Calculator

Create an object-oriented program that performs calculations on a rectangle.

Console

```
Rectangle Calculator

Height:    10
Width:     20
Perimeter: 60
Area:      200
* * * * *
*                               *
*                               *
*                               *
*                               *
*                               *
*                               *
*                               *
*                               *
* * * * *

Continue? (y/n): y

Height:     5
Width:      10
Perimeter:  30
Area:       50
* * * * *
*                               *
*                               *
*                               *
* * * * *

Continue? (y/n): n

Bye!
```

Specifications

- Use a Rectangle class that provides attributes to store the height and width of a rectangle. This class should also provide methods that calculate the perimeter and area of the rectangle. In addition, it should provide a method that gets a string representation of the rectangle.
- When the program starts, it should prompt the user for height and width. Then, it should create a Rectangle object from the height and width and use the methods of that object to get the perimeter, area, and string representation of the object.

Project 14-2: Card Dealer

Create an object-oriented program that creates a deck of cards, shuffles them, and deals the specified number of cards to the player.

Console

```
Card Dealer

I have shuffled a deck of 52 cards.

How many cards would you like?: 7

Here are your cards:
Jack of Hearts
Jack of Diamonds
2 of Diamonds
6 of Spades
Jack of Spades
6 of Hearts
King of Diamonds

There are 45 cards left in the deck.

Good luck!
```

Specifications

- Use a Card class to store the rank and suit for each card. In addition, use a method to get a string representation for each card such as “Ace of Spades”, “2 of Spades”, etc.
- Use a Deck class to store the 52 cards in a standard playing deck (one card for each rank and suit):
ranks: 2, 3, 4, 5, 6, 7, 8, 9, 10, Jack, Queen, King, Ace
suits: Clubs, Diamonds, Hearts, Spades
- The Deck class should include a method that shuffles the deck, a method that counts the number of cards in the deck, and a method that deals a card from the deck, which should reduce the count of the cards in the deck by 1.
- When the program starts, it should get a new deck of cards, shuffle them, and display a message that indicates the total number of cards in the deck. To shuffle the cards, you can use the shuffle function of the random module described in chapter 6.
- The program should prompt the user for the desired number of cards. Then, it should deal the user the desired number of cards and display a message that indicates the number of cards left in the deck.

Project 14-3: Customer Viewer

Create an object-oriented program that reads a list of Customer objects from a CSV file and allows the user to enter the data for a customer by specifying the customer's ID.

Console

```
Customer Viewer

Enter customer ID: 103

Art Venere
8 W Cerritos Ave #54
Bridgeport, NJ 08014

Continue? (y/n): y

Enter customer ID: 104

Lenna Paprocki
Feltz Printing
639 Main St
Anchorage, AK 99501

Continue? (y/n): y

Enter customer ID: 99

No customer with that ID.

Continue? (y/n): n

Bye!
```

Specifications

- Your instructor should provide a CSV file named `customers.csv` that contains customer data.
- Use a `Customer` class to store the customer data. This class should include these attributes: `id`, `firstName`, `lastName`, `company`, `address`, `city`, `state`, `zip`.
- In addition, this class should include a property or method that returns the full address. This address should have three lines if the company attribute is empty or four lines if the company attribute is not empty.
- Create a function that reads the customer data from a CSV file and creates `Customer` objects from it.
- To find the customer with the specified ID, you need to loop through each `Customer` object in the list of `Customer` objects and check whether the specified ID matches the ID stored in the `Customer` object.
- If the specified ID isn't found, display an appropriate message.

Project 15-1: Rectangle or Square Calculator

Create an object-oriented program that uses inheritance to perform calculations on a rectangle or a square.

Console

```
Rectangle Calculator

Rectangle or square? (r/s): r
Height:      5
Width:       10
Perimeter:   30
Area:        50
* * * * *
*           *
*           *
*           *
* * * * *

Continue? (y/n): y

Rectangle or square? (r/s): s
Length:      5
Perimeter:   20
Area:        25
* * * * *
*           *
*           *
*           *
* * * * *

Continue? (y/n): n

Bye!
```

Specifications

- Use a Rectangle class that provides attributes to store the height and width of a rectangle. This class should also provide methods that calculate the perimeter and area of the rectangle. In addition, it should provide a `__str__` method that returns a string representation of the rectangle.
- Use a Square class that inherits the Rectangle class. This class should set the height and the width of the Rectangle superclass to the length that's set in the Square subclass.
- The program should determine whether the user wants to enter a rectangle or a square.
- For a rectangle, the program should get the height and width from the user.
- For a square, the program should get the length of the square from the user.

Project 15-2: Roshambo

Create an object-oriented program for a Roshambo game where the user can choose to compete against one of two computer players: Bart or Lisa.

Console

```
Roshambo Game

Enter your name: Joel

Would you like to play Bart or Lisa? (b/l): b

Rock, paper, or scissors? (r/p/s): r

Joel: rock
Bart: rock
Draw!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): p

Joel: paper
Bart: rock
Joel wins!

Play again? (y/n): y

Rock, paper, or scissors? (r/p/s): s

Joel: scissors
Bart: rock
Bart wins!

Play again? (y/n): n

Thanks for playing!
```

Specifications

- Create a class named Player that provides attributes for storing the player's name and Roshambo value.
- Create a class named Bart that inherits the Player class and adds a generateRoshambo method. This method should set the Roshambo attribute to rock.
- Create a class named Lisa that inherits the Player class and adds a generateRoshambo method. This method should randomly select rock, paper, or scissors.
- The program should allow the user (Player) to play Roshambo against Bart or Lisa.
- In the game of Roshambo, rock beats scissors, paper beats rock, and scissors beats paper.

Enhancement

- Keep track of wins and losses and display them at the end of each session.

Project 15-3: Customer or Employee Creator

Create an object-oriented program that allows you to enter data for customers and employees.

Console

```
Customer/Employee Data Entry

Customer or employee? (c/e): c

DATA ENTRY
First name: Frank
Last name: Wilson
Email: frank44@gmail.com
Number: M10293

CUSTOMER
First name: Frank
Last name: Wilson
Email: frank44@gmail.com
Number: M10293

Continue? (y/n): y

Customer or employee? (c/e): e

DATA ENTRY
First name: Joel
Last name: Murach
Email: joel@murach.com
SSN: 123-45-6789

EMPLOYEE
First name: Joel
Last name: Murach
Email: joel@murach.com
SSN: 123-45-6789

Continue? (y/n): n

Bye!
```

Specifications

- Create a Person class that provides attributes for first name, last name, and email address. This class should provide a property or method that returns the person's full name.
- Create a Customer class that inherits the Person class. This class should add an attribute for a customer number.
- Create an Employee class that inherits the Person class. This class should add an attribute for a social security number (SSN).
- The program should create a Customer or Employee object from the data entered by the user, and it should use this object to display the data to the user. To do that, the program can use the `isinstance()` function to check whether an object is a Customer or Employee object.

Project 15-4: Random Integer List

Create an object-oriented program that uses a custom list object to automatically generate and work with a series of random integers.

Console

```
Random Integer List

How many random integers should the list contain?: 12

Random Integers
=====
Integers:  17, 34, 34, 15, 71, 44, 97, 48, 19, 12, 83, 42
Count:      12
Total:      516
Average:    43.0

Continue? (y/n): y

Random Integers
=====
Integers:  52, 88, 10, 77, 56, 91, 17, 51, 22, 14, 48, 37
Count:      12
Total:      563
Average:    46.917

Continue? (y/n): n

Bye!
```

Specifications

- Create a `RandomIntList` class that inherits the list class. This class should allow a programmer to create a list of random integers from 1 to 100 by writing a single line of code. For example, a programmer should be able to create a custom list that stores 12 random integers with this line of code:

```
int_list = RandomIntList(12)
```

- To do that, you can use the `self` keyword to access the list superclass like this:

```
self.append(rand_int)
```
- The `RandomIntList` class should contain methods for getting the count, average, and total of the numbers in the list. In addition, it should contain a `__str__` method for displaying a comma-separated list of integers as shown above.
- The program should use the `RandomIntList` class to generate the list of random integers, display the list, and get the summary data (count, total, and average).

Project 16-1: Blackjack

Design and implement an object-oriented program for a simple game of blackjack that provides for one player and a dealer (the computer).

Console

```
Blackjack

DEALER'S SHOW CARD:
9 of Clubs

YOUR CARDS:
2 of Hearts
Queen of Clubs

Hit or stand? (hit/stand): hit

YOUR CARDS:
2 of Hearts
Queen of Clubs
7 of Clubs

Hit or stand? (hit/stand): stand

DEALER'S CARDS:
9 of Clubs
7 of Spades
Queen of Spades

YOUR POINTS:      19
DEALER'S POINTS:  26

Yay! The dealer busted. You win!

Play again? (y/n): n

Come back soon!
```

Specifications

- Design and implement the classes for the business tier.
- Use functions to implement the user interface tier.
- Store the code for each tier in its own file.
- If necessary, learn the rules of Blackjack by researching it on the web.
- Use a standard 52-card deck of playing cards.
- The dealer must continue taking cards until the dealer's hand has at least 17 points.
- Don't implement betting.
- Don't allow a player to "split" a hand or "double down."

Project 17-1: Customer Data Importer

Create a program that imports customer data from a CSV file into a database table.

Console

```
Customer Data Importer

CSV file:  customers.csv
DB file:   customers.sqlite
Table name: Customer

All old rows deleted from Customer table.
500 row(s) inserted into Customer table.
```

Specifications

- Your instructor should provide you with the CSV and database files shown above (customers.csv and customers.sqlite). The SQLite database file should contain a table named Customer.
- The program should begin by deleting any old data from the Customer table. Then, it should insert all data from the customers.csv file into the Customer table of the SQLite database.
- The CSV file should be in this format:

```
first_name,last_name,company_name,address,city,state,zip
James,Butler,,6649 N Blue Gum St,New Orleans,LA,70116
Josephine,Darakjy, ,4 B Blue Ridge Blvd,Brighton,MI,48116
Art,Venere,,8 W Cerritos Ave #54,Bridgeport,NJ,08014
Lenna,Paprocki,Feltz Printing,639 Main St,Anchorage,AK,99501
```

- The Customer table should have the following columns and data types:
- | | |
|-------------|---------------------|
| customerID | INTEGER PRIMARY KEY |
| firstName | TEXT |
| lastName | TEXT |
| companyName | TEXT |
| address | TEXT |
| city | TEXT |
| state | TEXT |
| zip | TEXT |
- This program must complete within a few seconds. If it takes longer than that, you need to figure out how to improve its speed.
 - Use SQLite to view the data and make sure that it has been added to the database correctly. In particular, check to make sure the database automatically generates the customer IDs.

Project 17-2: Player Manager

Create a program that allows you to store the data for players of a game.

Console

```
Player Manager

COMMAND MENU
view - View players
add  - Add a player
del  - Delete a player
exit - Exit program

Command: view
Name      Wins    Losses    Ties    Games
-----
Mike      4        3         7       14
Joel      3        7        10       20

Command: add
Name: anne
Wins: 9
Losses: 5
Ties: 3
Anne was added to database.

Command: view
Name      Wins    Losses    Ties    Games
-----
Anne      9        5         3       17
Mike      4        3         7       14
Joel      3        7        10       20

Command: del
Name: anne
Anne was deleted from database.

Command: exit
Bye!
```

Specifications

- Your instructor should provide you with a database file (players_db.sqlite) that contains a Player table that stores the data for each player.
- Use the three-tier architecture (presentation, business, database) for this program, and store the code for each tier in a separate file.
- Display the players in order by wins, starting with the player with the most wins.
- Assume that the name for each player is unique.

Possible enhancement

- Add an “update” command. This command should prompt the user to enter the name of a player. Then, it should let the user update the wins, losses, and ties for the player.

Project 17-3: Task List

Create a program that allows you to manage a task list that's stored in a database.

Console

```
Task List

COMMAND MENU
view      - View pending tasks
history   - View completed tasks
add       - Add a task
complete  - Complete a task
delete    - Delete a task
exit      - Exit program

Command: view
1. Buy toothbrush
2. Do homework

Command: complete
Number: 2

Command: add
Description: Pay bills

Command: view
1. Buy toothbrush
2. Pay bills

Command: history
1. Get bike fixed (DONE!)
2. Call your mom (DONE!)
3. Do homework (DONE!)

Command: exit
Bye!
```

Specifications

- Your instructor should provide you with a database file (task_list_db.sqlite) that contains a Task table that stores the tasks.
- Use the three-tier architecture (presentation, business, database) for this program, and store the code for each tier in a separate file.
- The view command should only display tasks that have not been completed.
- The complete command should only mark a task as completed, not delete it from the database.
- The history command should allow you to view tasks that have been completed, but not deleted.

Project 17-4: Product Manager

Create a program that manages the products that are available from a guitar shop.

Console

```
Product Manager

CATEGORIES
Guitars | Basses | Drums

COMMAND MENU
view - View products by category
update - Update product price
exit - Exit program

Command: view
Category name: basses
Code      Name      Price
-----
precision Fender Precision 799.99
hofner     Hofner Icon    499.99

Command: update
Product code: hofner
New product price: 399.50
Product updated.

Command: view
Category name: basses
Code      Name      Price
-----
precision Fender Precision 799.99
hofner     Hofner Icon    399.50

Command: exit
Bye!
```

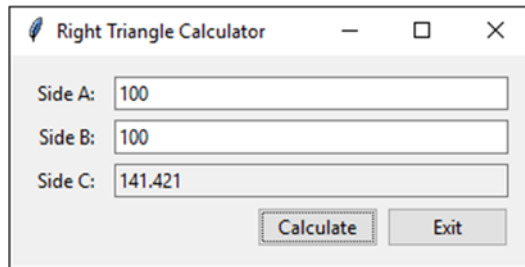
Specifications

- Your instructor should provide you with a database file (guitar_shop.sqlite) that contains Category and Product tables. These tables store the data for the categories and the products within each category.
- Use the three-tier architecture (presentation, business, database) for the program, and store the code for each tier in a separate file.
- Display the products alphabetically by product name.
- Assume that the name for each category is unique.
- Assume that the code for each product is unique.

Project 18-1: Right Triangle Calculator

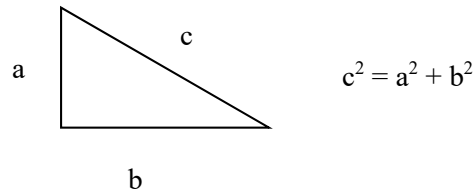
Create a GUI program that calculates the hypotenuse of a right triangle after the user enters the lengths of the two short sides and clicks the Calculate button.

GUI



Specifications

- Use the Pythagorean Theorem to calculate the length of the third side. The Pythagorean Theorem states that the square of the hypotenuse of a right-triangle is equal to the sum of the squares of the opposite sides:

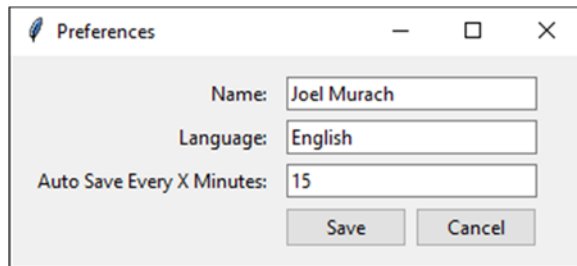


- As a result, you can calculate side C like this:
`c = square_root(a2 + b2)`
- Side C should be rounded to a maximum of 3 decimal places.

Project 18-2: Preferences

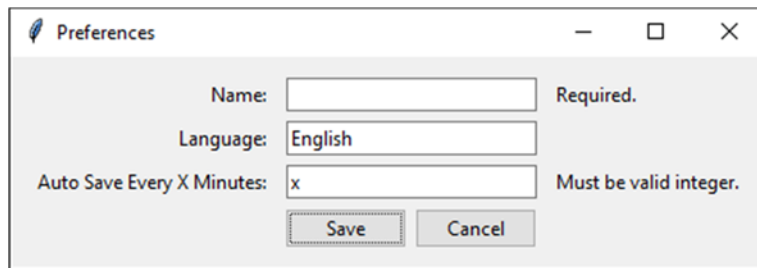
Create a GUI program that stores a user's preferences for a program.

GUI with valid data



A screenshot of a 'Preferences' dialog box. It has a title bar with a feather icon, the text 'Preferences', and standard window controls (minimize, maximize, close). The dialog contains three text input fields: 'Name:' with 'Joel Murach', 'Language:' with 'English', and 'Auto Save Every X Minutes:' with '15'. Below the fields are 'Save' and 'Cancel' buttons.

GUI with invalid data after clicking the Save button



A screenshot of the 'Preferences' dialog box after clicking the 'Save' button with invalid data. The 'Name' field is empty and has a 'Required.' message to its right. The 'Language' field contains 'English'. The 'Auto Save Every X Minutes' field contains 'x' and has a 'Must be valid integer.' message to its right. The 'Save' button is disabled (grayed out), while the 'Cancel' button remains active.

Specifications

- When the program starts, it should read the preferences from a file and display them in the GUI. If the program can't find the preferences file, it should display a blank name and standard default values of your choosing for language and auto save minutes.
- If the user enters valid data and clicks the Save button, the program should write the preferences to a file and close the GUI.
- If the user enters invalid data and clicks the Save button, the program should not save the data or close the GUI. Instead, it should display an appropriate message to the right of the text field as shown above.
- The Name and Language fields are required.
- The Auto Save field is required and must be a valid int value.
- If the user clicks the Cancel button, the program should close the GUI without saving any changes.

Note

- To display the validation messages, you can add a label to the third column after each text entry field. Then, you can use the same technique for setting text in this label as you do for text entry fields, and you can set the text to an empty string if there's no message for the field.