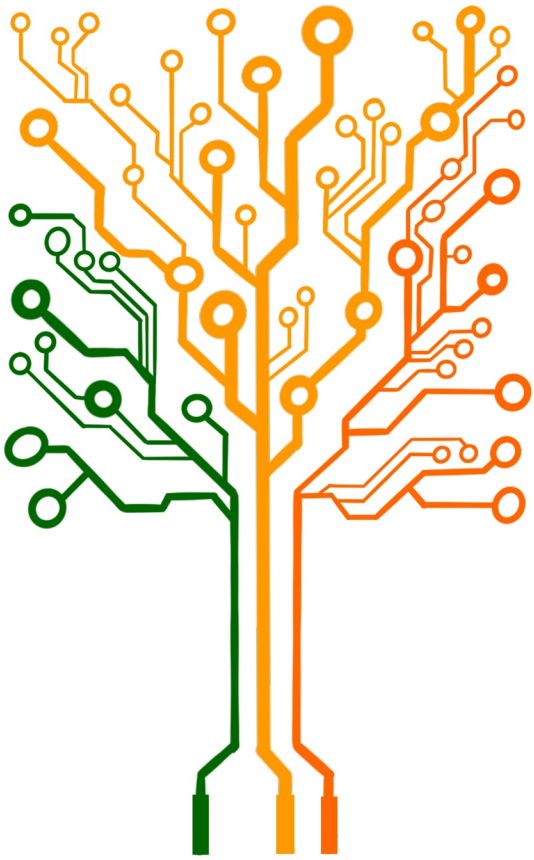
XU FEI ZHANG

Become proficient in Python

Learn Elementary School Math with Coding Learning Handbook



Learn elementary school math efficiently

Contains 60 Python programs covering all 49 units from third- to fifth-grade Chinese mathematics textbooks

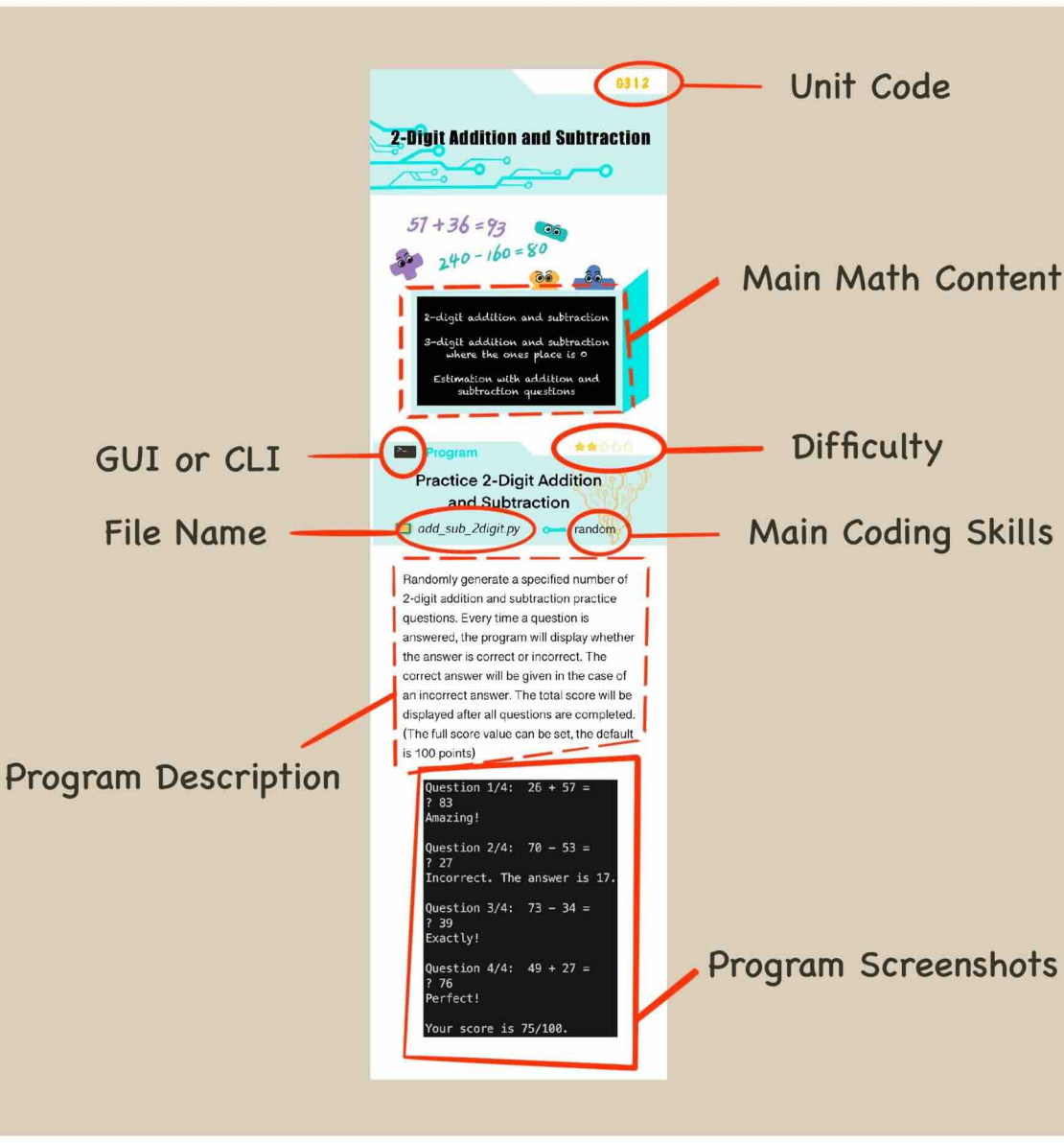
Handbook Introduction and Usage Information

This learning handbook contains descriptions for 60 Python programs (22 with graphical interfaces and 38 with command line interfaces). These programs cover all 49 units from the third- to fifth-grade Chinese mathematics textbooks of the People's Education Press. This approach of learning elementary math with coding allows learners to efficiently and effectively study mathematics while simultaneously becoming proficient in a programming language.

The “Learn Elementary School Math with Coding” project can serve as a supplementary learning resource for school mathematics curriculums or a way to accelerate the completion of elementary school mathematics. For a detailed introduction, please refer to the project website: https://feli10.github.io/math-coding.

For code download and usage instructions, please refer to the project’s GitHub repository: https://github.com/feli10/math-coding.

The page structure for each unit in the handbook is as shown in the following diagram:



Unit Identifier

The organization of the handbook is based on the elementary school Chinese mathematics textbooks of the People's Education Press. Each unit of this handbook is identified by a three-digit number starting with "G," for example, G311. The digits in the number have the following meanings:

- The "G" and the digit following it represent the grade.
- The second digit may be either 1 or 2. 1 represents the first semester, and 2 represents the second semester.
- The third digit represents the specific unit within the semester.

Therefore, G311 identifies the first unit of the first semester of third grade.

Program Description

Each unit contains 1-2 Python programs closely related to the math content of that unit. Each program description contains the following:

- File name.
- Graphical display or command line display.
- Difficulty rating (1 to 5 stars).
- Programming skills used.
- Program description.
- Screenshots of program output.

Contents

Grade 3 Semester 1

Grade 3 Semester 2

Contents

Grade 4 Semester 1

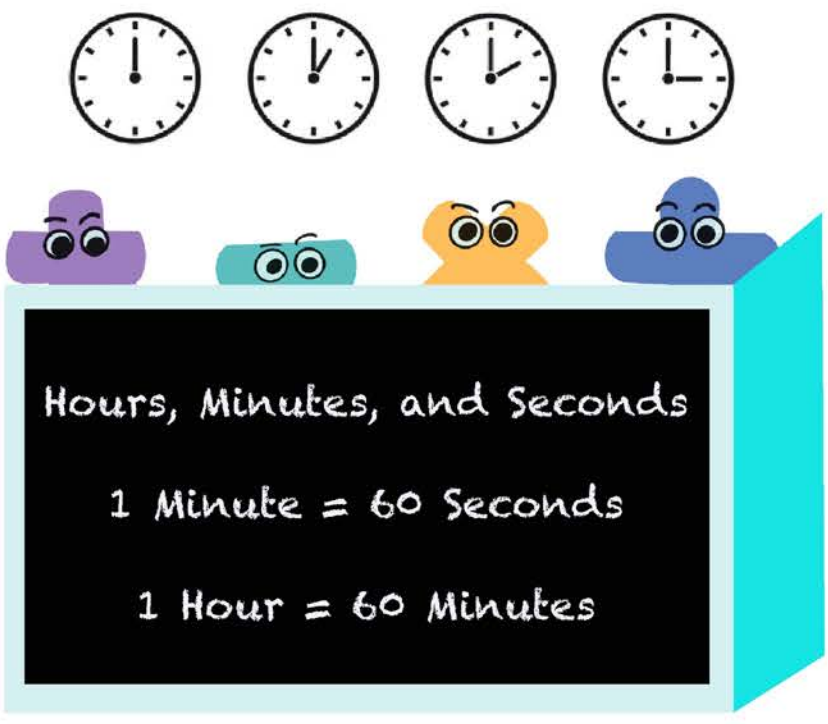
Grade 4 Semester 2

Contents

Grade 5 Semester 1

Grade 5 Semester 2

Telling Time - Hours, Minutes, and Seconds



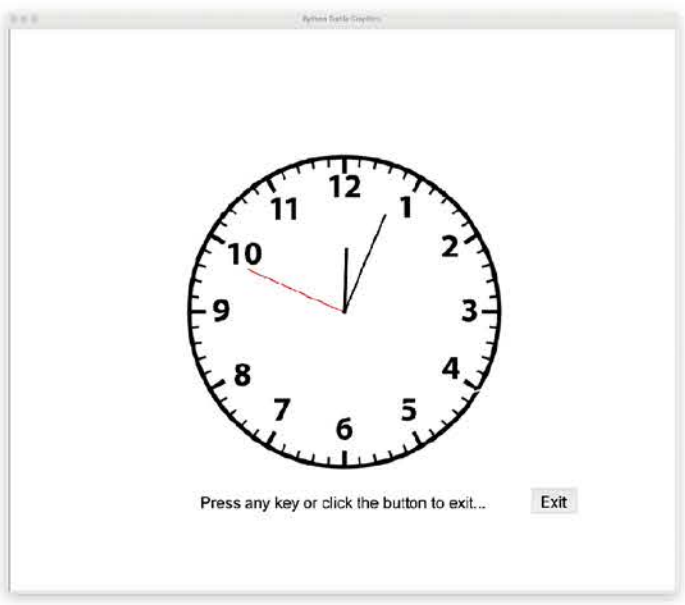
Program 1

Analog Clock

clock.py turtle, tkinter, exception

The first program simulates an analog clock. The hour, minute, and second hands move in real time when the program is running. The program uses the turtle and tkinter modules to achieve real-time animation in a graphical display.

When the program is running, press any button or click on the EXIT button to end the program. The program defaults to displaying the change in actual time. However, parameters can be adjusted to make the clock run faster or slower.



Program 2

Digital Clock

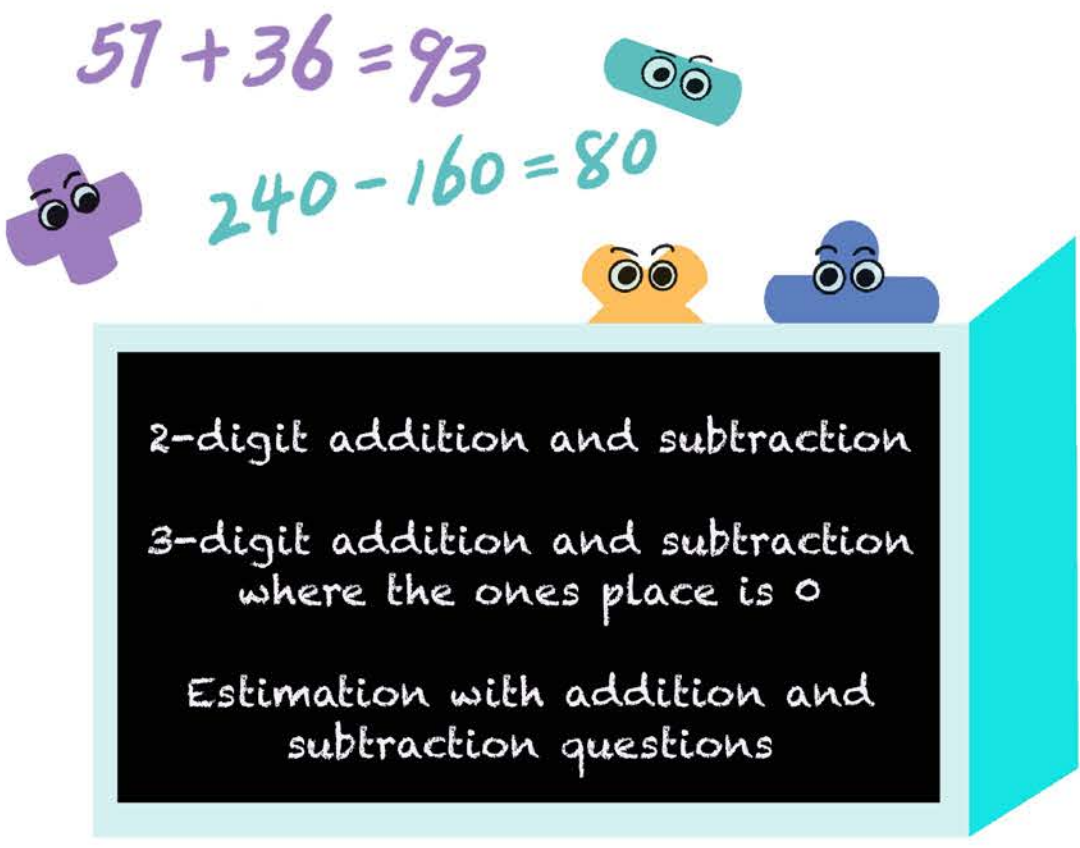
digital_clock.py exception

The second program creates a digital clock in the command line interface. The clock displays time in the format 00:00:00, where the digits for hours, minutes, and seconds change accordingly as time progresses.

The program allows users to set a specific time duration (in seconds) for the timer. The program will automatically end when the timer expires or can be manually terminated using Ctrl-C. By default, the timer runs in real-time. However, parameters can be adjusted to greatly speed up the time change in order to quickly examine changes in the minute and hour digits.

```
Please set a timer (in seconds): 80
00:01:20
Time is up!
```

2-Digit Addition and Subtraction



Program

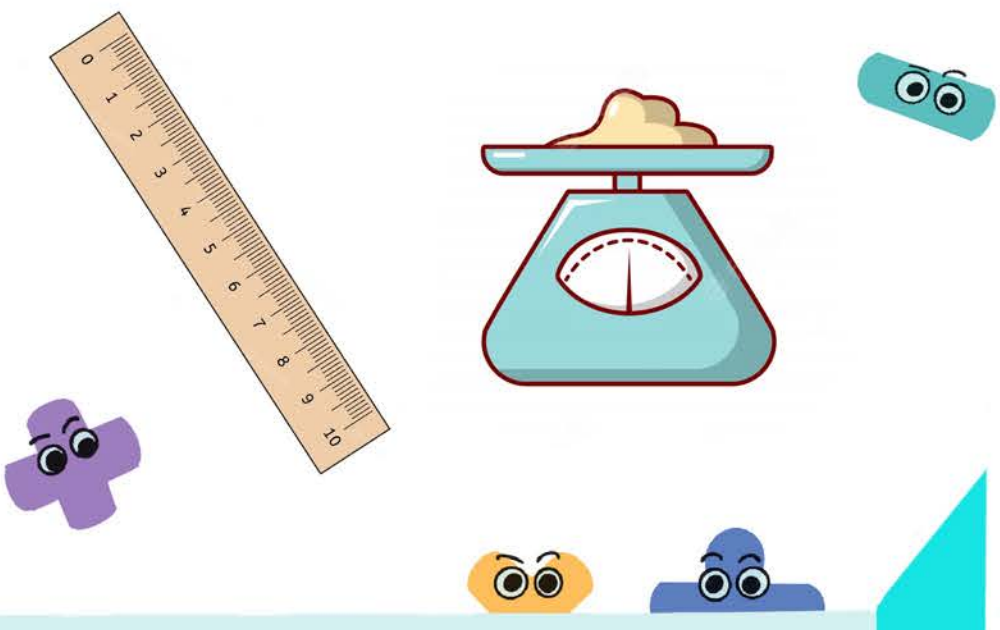
Practice 2-Digit Addition and Subtraction

add_sub_2digit.py random

Randomly generate a specified number of 2-digit addition and subtraction practice questions. Every time a question is answered, the program will display whether the answer is correct or incorrect. The correct answer will be given in the case of an incorrect answer. The total score will be displayed after all questions are completed. (The full score value can be set, the default is 100 points)

```
Question 1/4:  26 + 57 =
? 83
Amazing!

Question 2/4:  70 - 53 =
? 27
Incorrect. The answer is 17.

Question 3/4:  73 - 34 =
? 39
Exactly!

Question 4/4:  49 + 27 =
? 76
Perfect!

Your score is 75/100.
```

Measurements



Units of length:
millimeter, centimeter, decimeter, meter, kilometer

1 cm = 10 mm

1 dm = 10 cm

1 m = 10 dm

1 km = 1000 m

Units of mass:
gram, kilogram, ton

1 kg = 1000 g

1 t = 1000 kg

Program

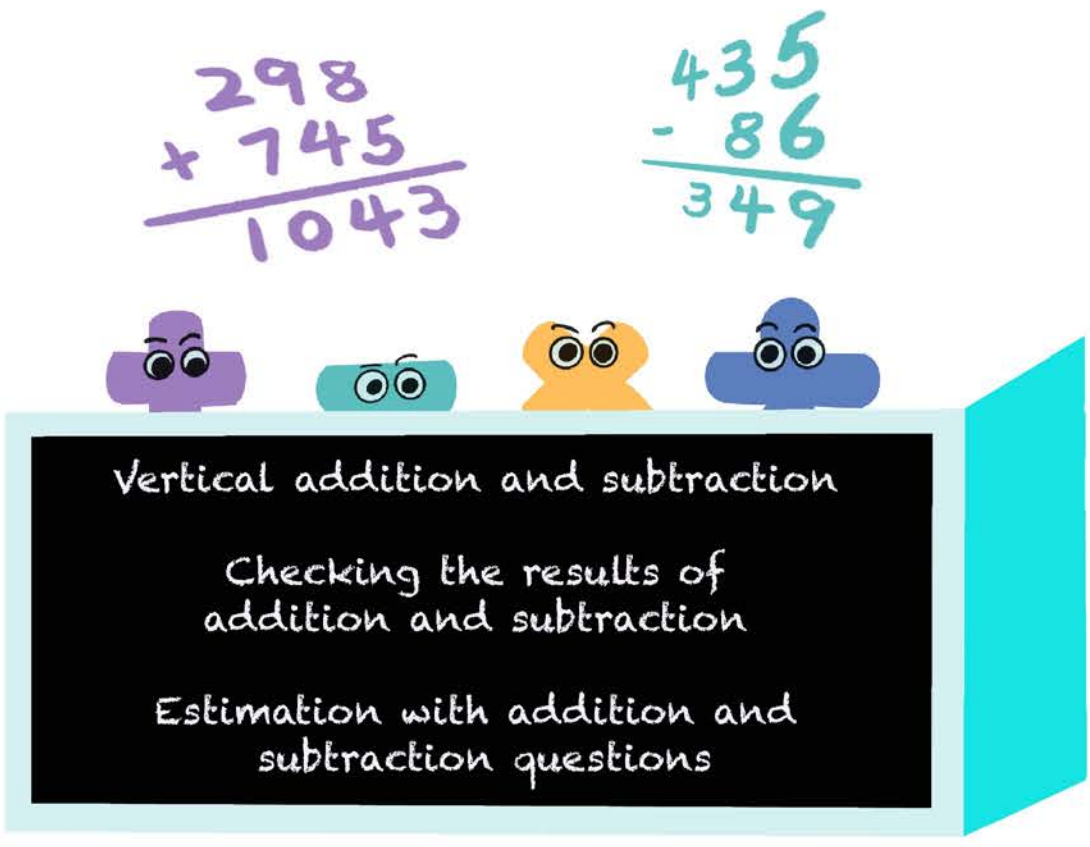
Practice Unit Conversion

unit_conversion.py random

Randomly generate a specified number of length and mass unit conversion questions. Since fractions and decimals have not yet been learned, the questions are all unit conversions from larger units to smaller units. For example, questions like 1 dm = __cm will appear, but 1cm = __dm will not. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set, the default is 100 points).

```
Question 1/4:  1t = __kg
? 10
Incorrect. The answer is 1000.

Question 2/4:  1kg = __g
? 1000
Great!

Question 3/4:  1m = __cm
? 100
Bingo!

Question 4/4:  1dm = __mm
? 100
Fabulous!

Your score is 75/100.
```

Vertical Addition and Subtraction



>_ Program 1

Vertical Addition

add_vertical.py string

This program sums two natural numbers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of addition instead of using the programming language's built-in "+" operator to get the result directly. This way, learners can strengthen their understanding and mastery of vertical addition through programming.

```
Enter the first number: 298
Enter the second number: 745
    2 9 8
+   7 4 5
---------
  1 0 4 3
```

>_ Program 2

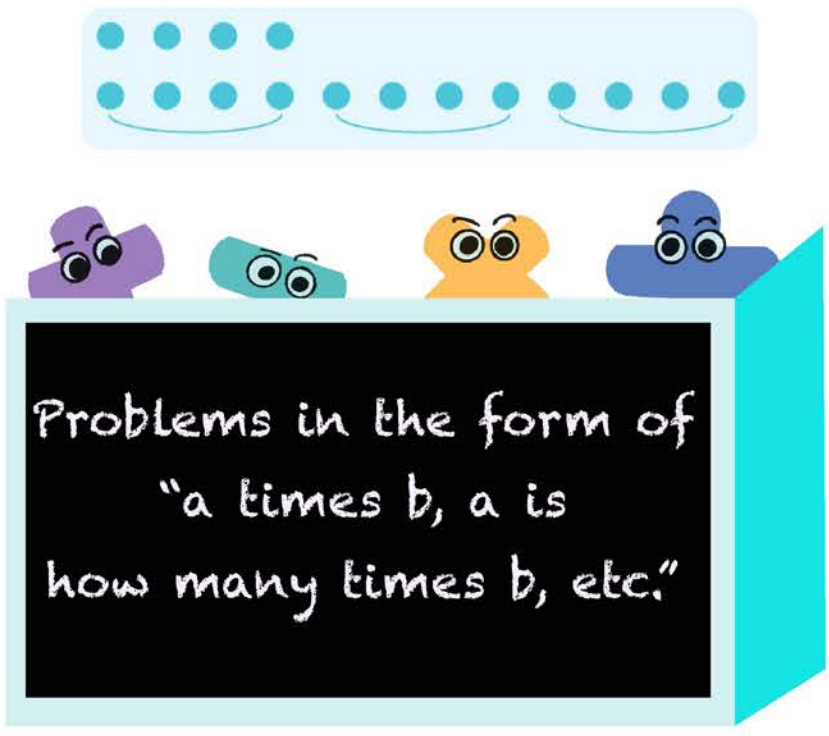
Vertical Subtraction

sub_vertical.py string

This program subtracts two natural numbers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of subtraction instead of using the programming language's built-in "-" operator to get the result directly. This way, learners can strengthen their understanding and mastery of vertical subtraction through programming.

```
Enter the first number: 435
Enter the second number: 86
  4 3 5
-   8 6
-------
  3 4 9
```

Multiplication Word Problems



Program

Practice Multiplication Word Problems

a_times_b.py random

Randomly generate a specified number of multiplication word problems. The problems are of three simple types, examples of each are as follows:

- What is 3 times 4?
- 12 is how many times 4?
- 3 times a number is 12; what is this number?

Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set, the default is 100 points).

```
Question 1/4:  What is 5 times 4?
? 20
Well done!

Question 2/4:  3 times a number is 6, what is this number?
? 18
Incorrect. The answer is 2.

Question 3/4:  What is 3 times 6?
? 18
Amazing!

Question 4/4:  56 is how many times 8?
? 7
Super!

Your score is 75/100.
```

Short Multiplication

Mental multiplication of multi-digit numbers by a single-digit number

Vertical multiplication of multi-digit numbers by a single-digit number

Multiplying by 0

Multiplication estimation using the approximate symbol

Two-step multiplication and division word problems

Program

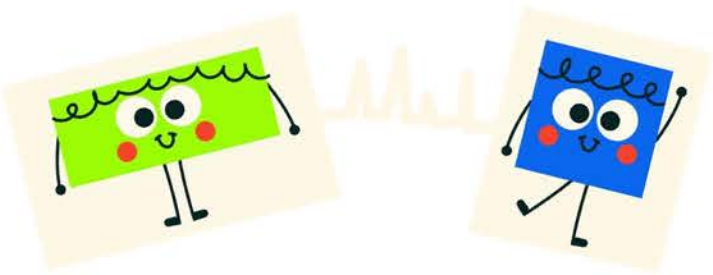
Short Multiplication

short_multiplication.py string

This program multiplies two natural numbers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of multiplication instead of using the programming language's built-in "*" operator to get the result directly. This way, learners can strengthen their understanding and mastery of vertical multiplication through programming.

```
Enter the first number: 343
Enter the second number (1-digit): 6
    3 4 3
x       6
---------
  2 0 5 8
```

Rectangles and Squares



Quadrilaterals (polygons with 4 sides)

Rectangles and squares

Perimeter of rectangles and squares

Perimeter of rectangle = (length + width) × 2

Perimeter of square = edge length × 4

Program

Create Rectangle Class

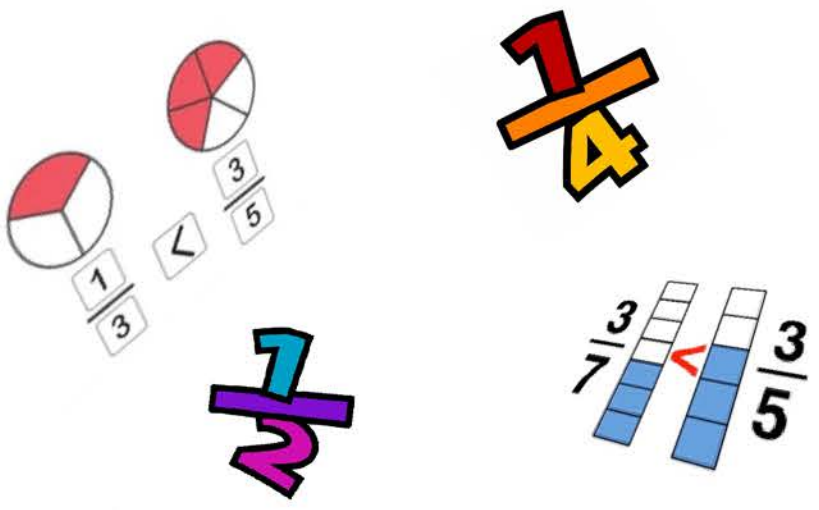
rectangle.py class

This program creates the Rectangle class. After instantiating a rectangle object, you can access its length and width, calculate its perimeter, and judge whether it is a square. You can also use the print() function to display relevant information about the rectangle object.

The program's logic is not complicated. Its main purpose is to use simple concepts such as the length, width, and perimeter rectangles as an example to help learners understand the concepts of classes and objects, as well as to gain a preliminary understanding of object-oriented programming (OOP).

```
>>> rect1 = Rectangle(5, 2)
>>> print(rect1)
Rectangle
length: 5
width: 2
perimeter: 14

>>> rect2 = Rectangle(3)
>>> print(rect2)
Square
side: 3
perimeter: 12

>>> rect2.length = 4
>>> print(rect2)
Rectangle
length: 4
width: 3
perimeter: 14

>>>
```

A Preliminary Understanding of Fractions



Program

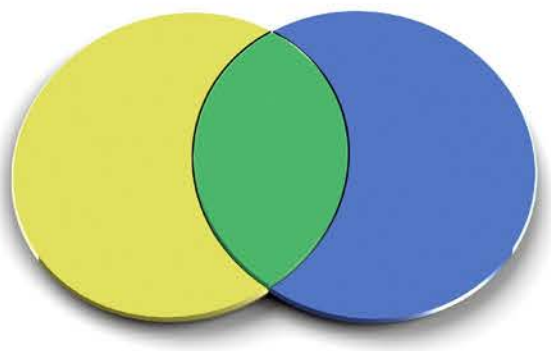
Comparing Fractions

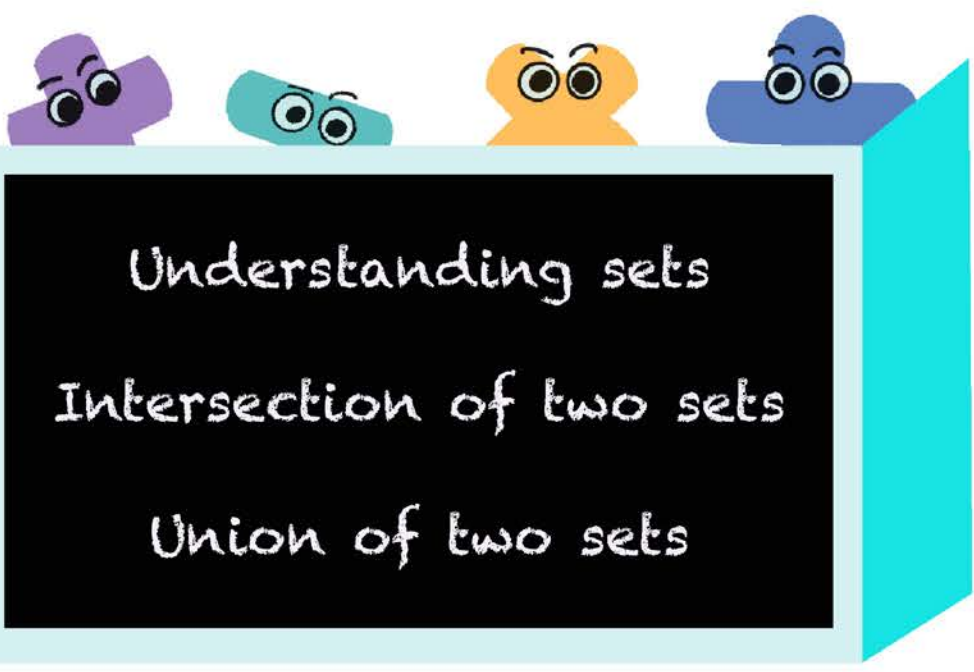
compare_fractions.py random

Randomly generate a specified number of practice questions on comparing fractions with the same numerator or denominator. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set, the default is 100 points).

```
Question 1/4:  2/3 __ 1/3
(> or <) ? >
Great!

Question 2/4:  5/9 __ 5/6
(> or <) ? <
Excellent!

Question 3/4:  2/4 __ 2/6
(> or <) ? <
Incorrect. The answer is >.

Question 4/4:  3/9 __ 5/9
(> or <) ? <
Good!

Your score is 75/100.
```

Sets





Program

Set Operations

sets.py set, random

The program uses two methods to obtain the intersection and union of two sets: the first is to use the list data type to represent the set, and program the intersection and union operations according to the definition (still in the form of list); the second is to use Python's built-in set data type and operations to directly obtain the intersection and union of two sets.

The program has two main purposes. Firstly, it lets learners realize, through random examples, that the sum of the elements of two sets minus the number of common elements equals the number of all elements. Secondly, it introduces learners to the set data type, which is especially useful for set operations. Unlike lists, sets don't contain repeating elements, and their elements aren't ordered.

```
My set operations using list:
A: [6, 4, 1, 2], 4 elements.
B: [8, 6, 7, 1, 5], 5 elements.
Intersection: [6, 1], 2 elements.
Union: [6, 4, 1, 2, 8, 7, 5], 7 elements.
4 + 5 - 2 = 7

Built-in set operations:
A: {1, 2, 4, 6}, 4 elements.
B: {1, 5, 6, 7, 8}, 5 elements.
Intersection: {1, 6}, 2 elements.
Union: {1, 2, 4, 5, 6, 7, 8}, 7 elements.
4 + 5 - 2 = 7
```

Orientations

North, South, East, West

Northeast, Northwest

Southeast, Southwest

Program

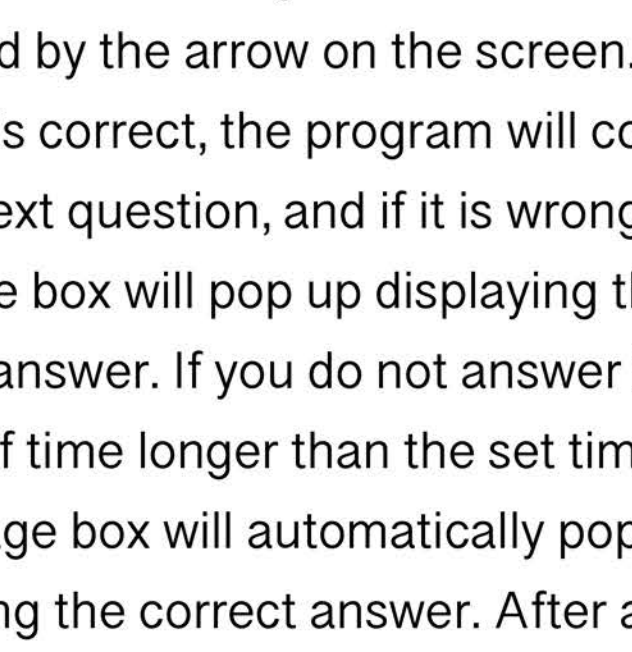
Practice Identifying Orientations

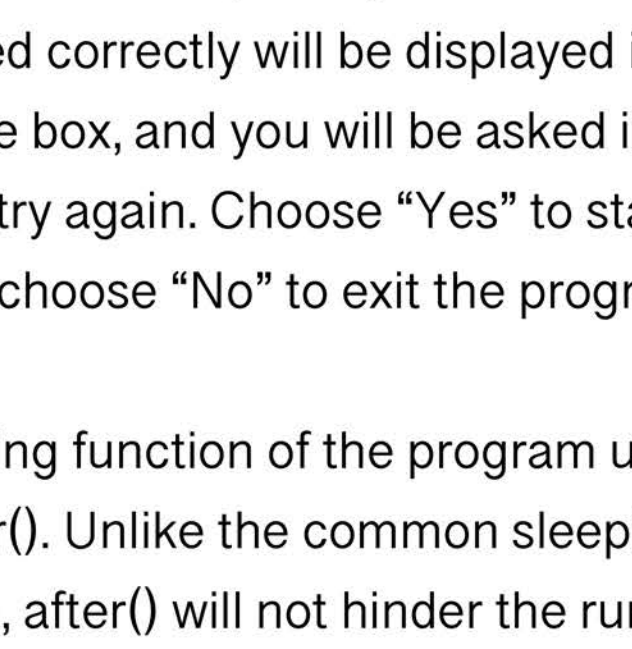
orientation.py tkinter, turtle, threads

Randomly generate a specified number of direction-identifying practice questions. the program adds a time limit to answering the questions in order to add some urgency and fun. Users can adjust the time limit or disable it as needed.

When the program is running, learners answer questions by choosing one of eight direction buttons (East, West, South, North, Southeast, Southwest, Northeast, and Northwest) according to the direction indicated by the arrow on the screen. If the answer is correct, the program will continue to the next question, and if it is wrong, a message box will pop up displaying the correct answer. If you do not answer for a period of time longer than the set time limit, a message box will automatically pop up, displaying the correct answer. After all questions are completed, the number answered correctly will be displayed in a message box, and you will be asked if you want to try again. Choose “Yes” to start over, or choose “No” to exit the program.

The timing function of the program uses tk's after(). Unlike the common sleep() function, after() will not hinder the running of the main thread of the program.





Short Division

Mental division with single-digit divisor

Vertical division with single-digit divisor

Division with remainder

Checking division results

Dividend is 0

Estimating division problems with single-digit divisor

Program

Short Division

short_division.py string

This program divides a multi-digit natural number by a one-digit non-zero natural number (both inputted by the user) and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of division instead of using the programming language's built-in "//" and "%" operators to get the result directly. This way, learners can strengthen their understanding and mastery of vertical division through programming.

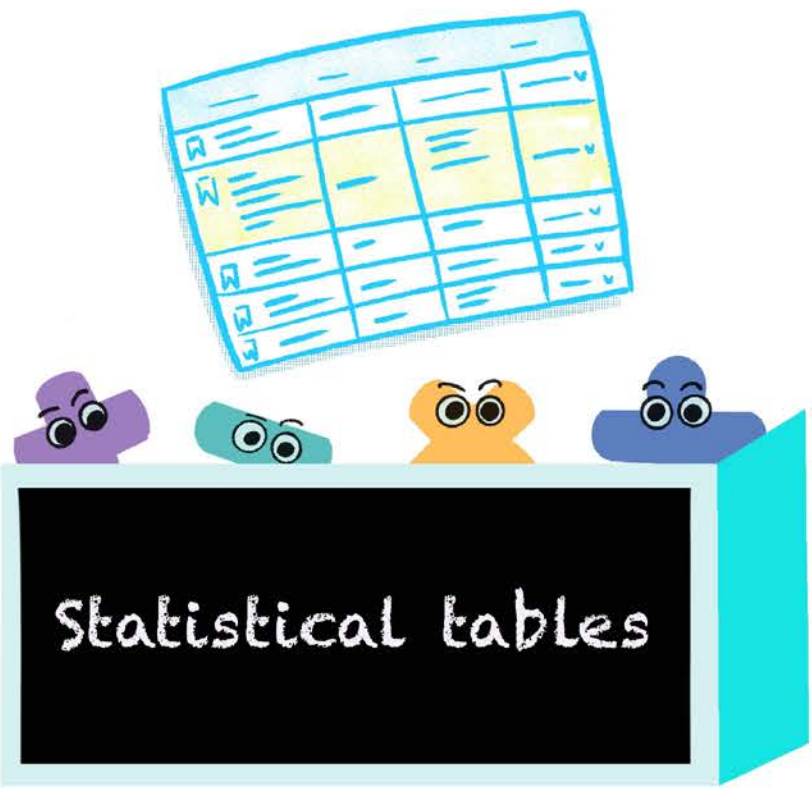
The most complicated part of this program is displaying vertical division. Unlike vertical addition and subtraction, which only has three rows, the number of rows of vertical division is not fixed, and the starting position of each row is constantly changing. When determining the starting position of each row, the following two factors are considered:

1. Vertical division contains several steps of "small" division, and the remainder of each step will be the highest part of the dividend in the next step. So, the digit difference between the dividend and the remainder in the small division of one step is the indentation of the dividend in the next.
2. If the remainder of a step of “small” division is zero, there will be at least one leading zero in the dividend of the next step. The leading zeros should be removed when displaying, but their positions should be kept as indentations.

```
Enter a natrual number as the dividend: 2160
Enter a 1-digit non-zero natrual number as the divisor: 2

  1080
  ----
2/2160
  2
  ----
   16
   16
  ----
    0

2160 / 2 = 1080 ... 0
```

```
Enter a natrual number as the dividend: 3501
Enter a 1-digit non-zero natrual number as the divisor: 5

   700
  ----
5/3501
  35
  ----
     1

3501 / 5 = 700 ... 1
```

Tables



Program

Creating and Displaying Tables

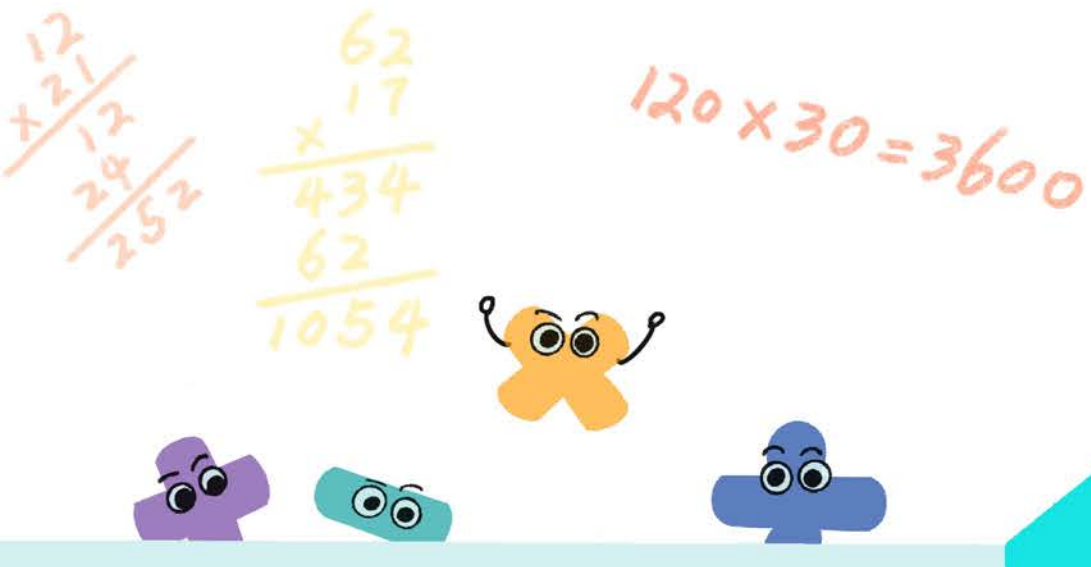
table.py class, list, string

The program creates a “table” class, which includes parameters such as row and column headers, row and column numbers, and the table’s data (specified or random). The class also includes methods such as data clearing and deleting rows or columns. The program lets learners familiarize themselves with statistical tables, understand the creation and use of classes and objects, and experience object-oriented programming (OOP, Object-Oriented Programming).

One of the main functions of the program is to display a table in the command line interface. Chinese and Western characters can be used in the table’s data. The width of the cell will be automatically adjusted according to the table’s data, and the row and column headers and data will be centered within their cells.

```
>>> vegetables = ['Tomato', 'Carrot', 'Cucumber', 'Corn']
>>> classes = ['Class1', 'Class2', 'Class3']
>>> table = Table(row_headers=classes, col_headers=vegetables)
>>> table.random()
>>> print(table)

|----------|----------|----------|----------|----------|
|          |  Tomato  |  Carrot  | Cucumber |   Corn   |
|----------|----------|----------|----------|----------|
|  Class1  |    18    |    8     |    4     |    14    |
|----------|----------|----------|----------|----------|
|  Class2  |    9     |    8     |    13    |    13    |
|----------|----------|----------|----------|----------|
|  Class3  |    4     |    17    |    3     |    9     |
|----------|----------|----------|----------|----------|

>>> print(table.shape)
3 x 4
>>> table.del_row(1)
>>> table.del_col(2)
>>> print(table)

|--------|--------|--------|--------|
|        | Tomato | Carrot |  Corn  |
|--------|--------|--------|--------|
| Class1 |   18   |    8   |   14   |
|--------|--------|--------|--------|
| Class3 |    4   |   17   |    9   |
|--------|--------|--------|--------|

>>> print(table.shape)
2 x 3
>>>
```

2-Digit Long Multiplication



Mental multiplication of multi-digit numbers with trailing zeros

Vertical multiplication of two two-digit numbers

Two-step multiplication and division word problems

Program

Long Multiplication 1

long_multiplication1.py string

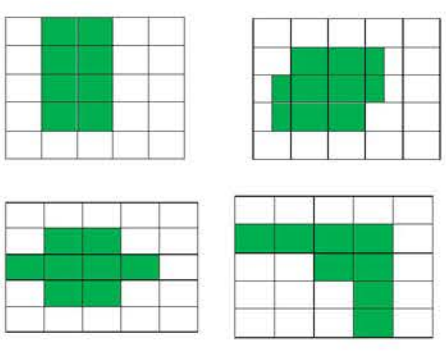
This program multiplies two natural numbers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of multiplication instead of using the programming language's built-in "*" operator to get the result directly.

The operation process of multi-digit vertical multiplication can be divided into two steps. The first step is multiplying multi-digit numbers by one-digit numbers. The second step is adding the products obtained in the first step. This program completes the first step by calling the previously written program short_multiplication.py (G36).

long_multipy_core(), which simulates the process of long multiplication, is separated from long_multiply(), which mainly displays long multiplication columns, so that long_multipy_core() can be reused in Long Multiplication 2, which is the final program of this multiplication series.

```
Enter the first number: 48
Enter the second number: 37
      4 8
x     3 7
---------
    3 3 6
  1 4 4
---------
  1 7 7 6
```

Area



The concept of area

Area units (cm^2, dm^2, m^2)

Area of rectangle and square

Area of rectangle = length × width

Area of a square = side length × side length

Conversion factors of area units

1 square decimeter = 100 square centimeters

1 square meter = 100 square decimeters

Tile laying word problems

Program

Drawing and Analyzing Rectangles

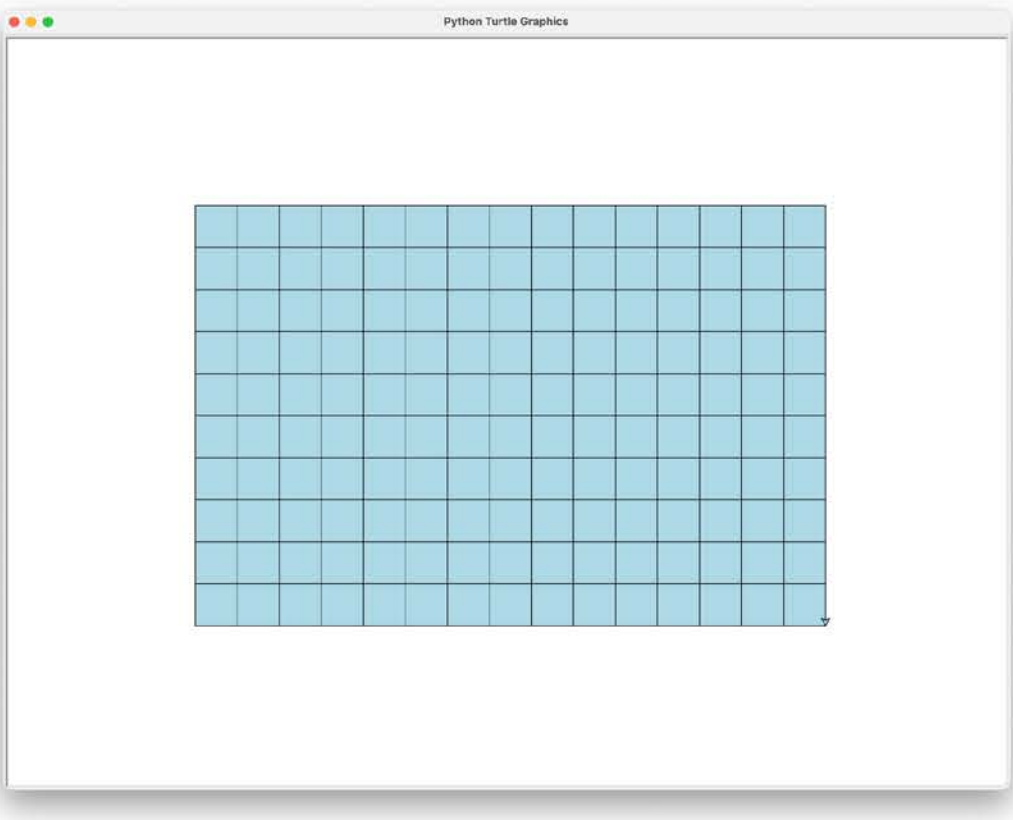
draw_rectangle.py tkinter, turtle, class, coordinate

This program expands upon the functionality of the previously written program rectangle.py (G317) by adding methods for drawing a rectangle and finding its area. To enhance the intuitiveness of the concept of area, you can choose to include a square grid when drawing a rectangle. Each square represents a unit area, and the number of squares is equal to the area of the rectangle.

Turtle and Tk are the two most commonly used tools in Python for programming graphic user interfaces. Turtle is simple and intuitive - the drawing process comes with animations, and you don't need to know plane coordinates to start using it; It is suitable for beginners. Tk is richer and has more powerful functions. It can not only draw pictures but can also be used to write application software under a graphical user interface. This program creates two methods, using Turtle and Tk, respectively, to draw rectangles. This allows learners to experience the functionalities of both tools, helping inform their choices when programming graphical interfaces in the future.

This program requires learners to have some basic understanding of coordinates. It should be noted that Turtle's coordinate origin (0, 0) is at the center of the screen (similar to Scratch), whereas Tk's coordinate origin (0, 0) is at the upper left corner (where down is the positive direction). Furthermore, because "width" and "height" are generally used to represent the horizontal and vertical dimensions of computer screens, the previous “length” and “width” attributes of the rectangle class have been changed to “width” and “height”.

```
>>> rect = Rectangle(30)
>>> print(rect)
Square
side: 10
perimeter: 40
area: 100

>>> rect.width = 40
>>> print(rect)
Rectangle
width: 15
height: 10
perimeter: 50
area: 150

>>> rect.draw(fill='lightblue', grid=True)
```



Years, Months, and Days

Days per month

Leap years

24-hour timekeeping

Time intervals

Program

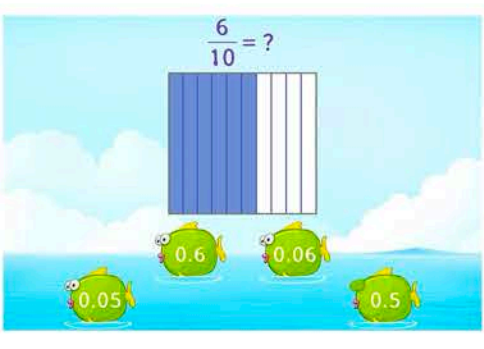
Display Calendar

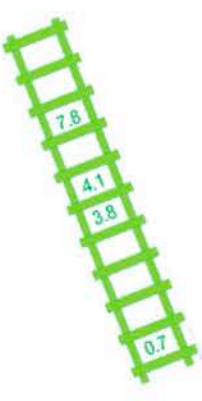
calendar.py string

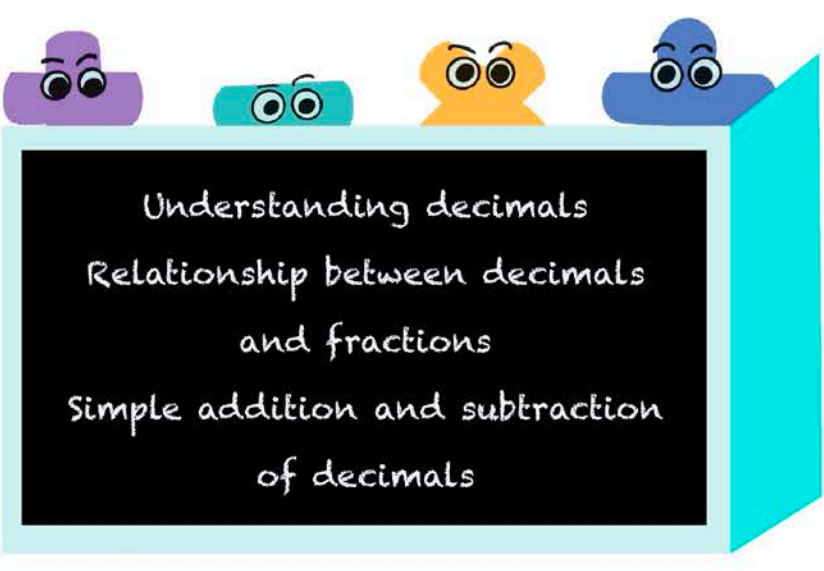
The user enters the year and month, and the program displays the calendar for that month on the screen.

```
Enter a year (1-9999): 2000
Enter a month (1-12): 2

   S   M   T   W   T   F   S
               1   2   3   4   5
   6   7   8   9  10  11  12
  13  14  15  16  17  18  19
  20  21  22  23  24  25  26
  27  28  29
```

Understanding Decimals







Program 1

Decimal Practice 1

decimal_practice1.py random

Randomly generate a specified number of decimal practice questions. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set; the default is 100 points). There are four types of questions - users can practice one type at a time or mix questions from multiple types. The four question types are as follows:

1. Converting fractions to decimals: The denominators of the fractions can be 10, 100, or 1000, with larger denominators being less likely to appear.
2. Converting decimals to fractions: Decimals are between 0 and 1 with 1-3 decimal places, with a lower likelihood for a larger number of decimal places. Since simplification of fractions has yet to be covered, there is no need to reduce them to the simplest form. As long as the fraction and the decimal are equal in value, the answer is considered correct. For example, converting 0.8 to 8/10 is acceptable.
3. Two-way unit conversions: Previously, in “Practice Unit Conversion” (G313), the focus was on converting from larger units to smaller units. In this program, two-way unit conversion questions are included, allowing you to input decimals or fractions when converting from smaller units to larger units. For example, "1 cm = 0.01 m" or "1 cm = 1/100 m." In addition to length and mass units, this program extends unit conversion to include currency units and area units.
4. Simple decimal addition and subtraction: These are one-digit decimal addition and subtraction questions within the range of 0 to 10. The operands may include integers to practice mixed operations with integers and decimals.

Through these exercises, you can gain a deeper understanding of the relationship between fractions and decimals and strengthen your comprehension of decimals as you transition from integer arithmetic to decimal arithmetic.

```
Question 1/4:  3/10
Decimal? 0.3
Impressive!

Question 2/4:  0.48
Fraction? 48/100
Awesome!

Question 3/4:  8 - 3.9 =
? 4.1
Incredible!

Question 4/4:  1cm² = __dm²
? 0.1
Incorrect. The answer is 0.01 or 1/100.

Your score is 75/100.
```

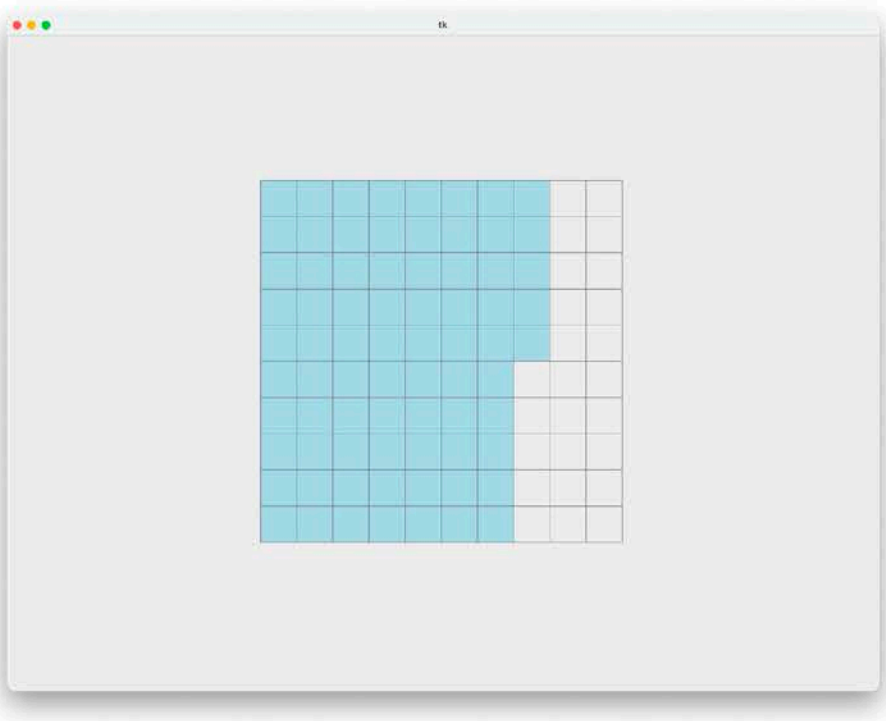
Program 2

Visualization of Decimals

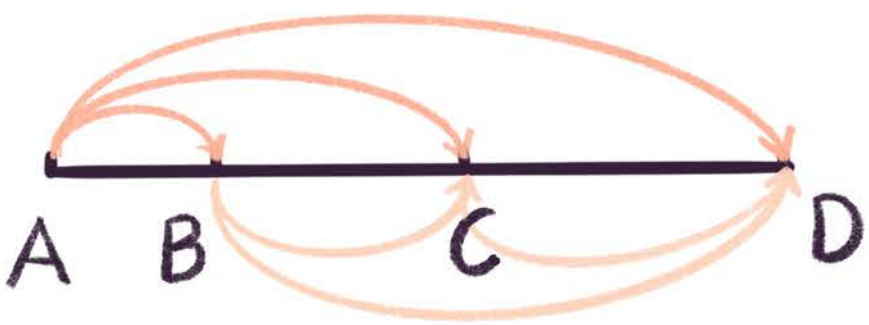
decimal_representation.py tkinter, coordinate

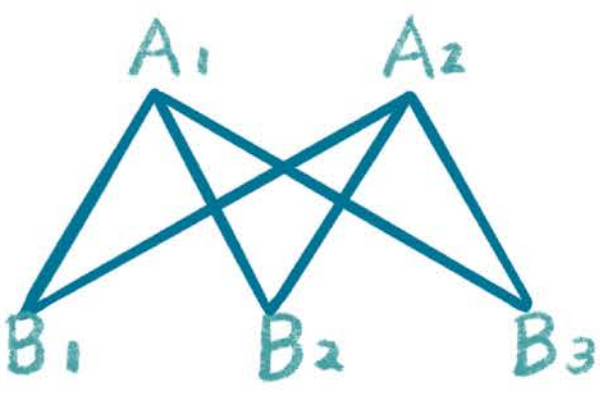
The user inputs a decimal or fraction between 0 and 1, and the program colors the portion of a square corresponding to this number. This visually intuitive representation helps to deepen the understanding of decimals and their connection to fractions.

```
Enter a decimal or fraction between 0 and 1: 0.75
```



Combinations





Counting principle

Choose one from each group

Select several from a group in order

Select several from a group without order

Program

Three Common Counting Problems

combination.py itertools, nested loop

This program provides solutions to three common counting problems using nested loops. The three types of counting problems are:

1. Cartesian Product: Select one item each from two groups. Examples include pairing tops with pants or selecting a main course and a side dish.
2. Permutation: Select and arrange two items from a group (order matters). In this case, choosing A first and B second is different from choosing B first and A second.
3. Combination: Select two items from a group (order doesn't matter). In this case, choosing A first and B second is the same as choosing B first and A second

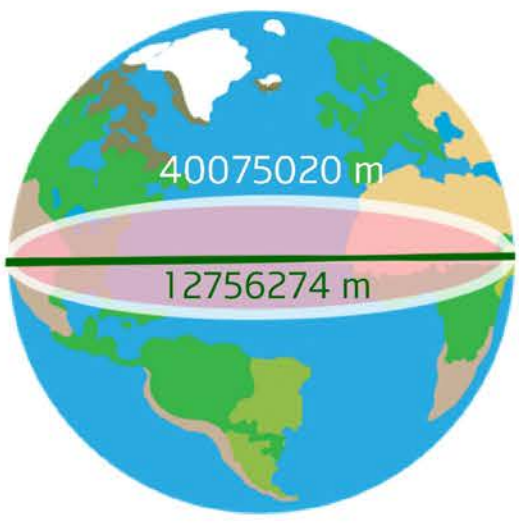
In each of these counting problems, listing all combinations without repetition is essential. This is where computer programs excel, as they can mechanically generate all scenarios using nested loops. The program also displays all combinations in an organized manner, making it easier for learners to identify patterns.

The program uses three slightly different types of nested loops to solve the above three types of problems. Selecting m items will use m nested loops, so when m is large, nested loops are not a good solution, and recursion is often used instead. However, when m is small, nested loops are the implementation that can best help learners understand counting problems.

Python's standard library provides the itertools module, which contains multiple functions that can be used directly to solve counting problems. The program compares the results of self-written functions to corresponding itertools functions, showing that they are the same.

```
Cartesian Product
A_:  Aa  Ab  Ac  Ad
B_:  Ba  Bb  Bc  Bd
C_:  Ca  Cb  Cc  Cd
D_:  Da  Db  Dc  Dd
There are 16 ways to select 1 item each from group1 (4 items) and group2 (4 items).

Permutation
A_:  AB  AC  AD
B_:  BA  BC  BD
C_:  CA  CB  CD
D_:  DA  DB  DC
There are 12 ways to select 2 items from group1 (4 items) when order matters.

Combination
A_:  AB  AC  AD
B_:  BC  BD
C_:  CD
D_:
There are 6 ways to select 2 items from group1 (4 items) when order doesn't matter.

>>> print(my_product_result == list(itertools.product(GROUP1, GROUP2)))
True
>>> print(my_permutation_result == list(itertools.permutations(GROUP1, 2)))
True
>>> print(my_combination_result == list(itertools.combinations(GROUP1, 2)))
True
>>>
```

Working with Large Numbers



Magnitudes
(thousands, millions, billions)

Unit conversions

Reading and writing large numbers

Comparing large numbers

Rounding to the nearest whole number

Natural numbers

Using a calculator

Program

Read Out Any Natural Number

read_number.py

string

The user can input any natural number less than 10 to the power of 48, and the program will display the number, read out in words.

```
Enter a natural number: 1234567
one million two hundred thirty four thousand five hundred sixty seven

Enter a natural number: 1,000,000,000,000
one trillion
```

Large Area Units

5100 72000 km²

Large area units: are (a), hectare (ha), and square kilometer (km²)

1 square kilometer = 100 hectare

1 hectare = 100 are

Program

Practice Area Unit Conversion

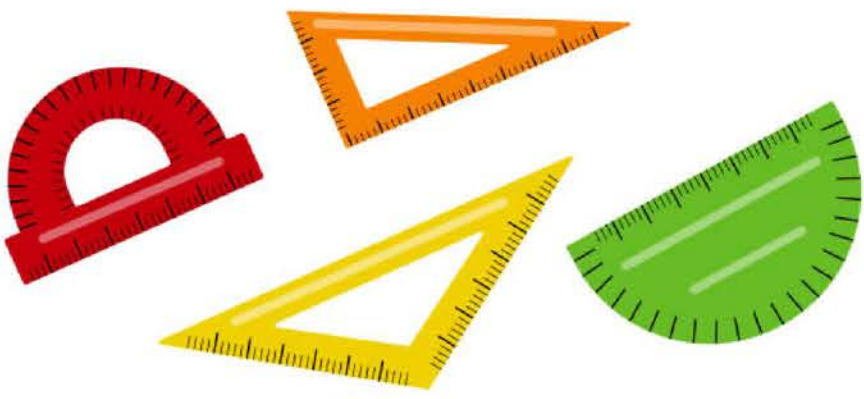
area_unit_conversion.py random

Randomly generate a specified number of area unit conversion questions. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set; the default is 100 points).

"Four Types of Decimal Practice Problems" (G327) includes a function that generates two-way unit conversion questions. This program expands on this function by adding the area units ares, hectares, and square kilometers to the pool of possible units for unit conversion questions. When converting from a smaller unit to a larger unit, you can enter either decimals or fractions (1 cm² = 0.0001 m² = 1/10000 m²). For answers that contain many digits, scientific notation can be used. For example, 1e3 represents 1 followed by three zeros, which is 1000; 1e-3 represents 1/(1e3), which is 1/1000 or 0.001.

```
Question 1/4:  1m² = __ha
? 0.01
Incorrect. The answer is 0.0001 or 1/10000.

Question 2/4:  1cm² = __dm²
? 1/100
Awesome!

Question 3/4:  1km² = __m²
? 1e6
Great!

Question 4/4:  1a = __km²
? 1e-4
Fantastic!

Your score is 75/100.
```

Measuring Angles



Line segment, line, ray

Angle, vertex, side

Right angle, straight angle, full angle (acute and obtuse angles were already covered in the second grade)

Measuring and drawing angles using a protractor

Program

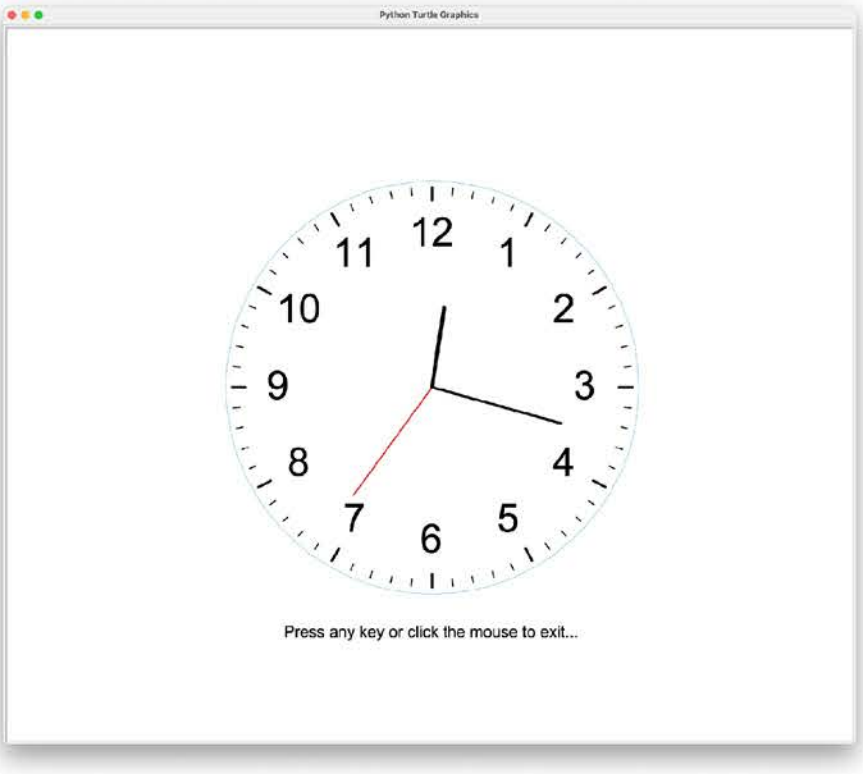
Draw Clock Dial

draw_clock_dial.py turtle, tkinter, threads

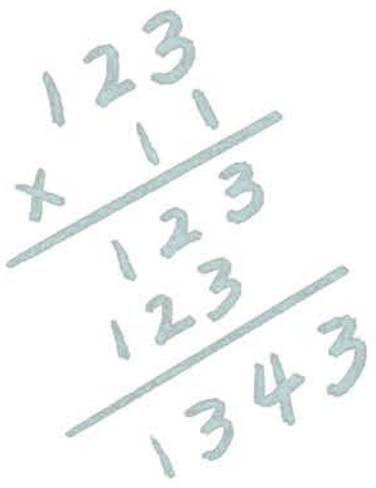
The program draws a clock dial using angles and lines. It has and has two main differences from the program "Analog Clock" (G311):

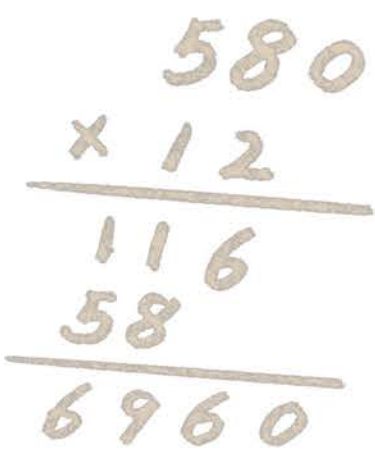
1. G311 uses a background image as its clock dial, whereas this program uses the Turtle module to draw the clock dial.
2. G311 uses the time module's sleep() for timing, whereas this program uses tk's after(). Unlike sleep(), after() does not block the main program thread.

After running the program, you can exit by pressing any key or by clicking inside the window. By adjusting parameters, you can also accelerate the movement of the clock hands or disable the animation for drawing the clock face.



3-Digit Long Multiplication





Program

Long Multiplication 2

long_multiplication2.py string

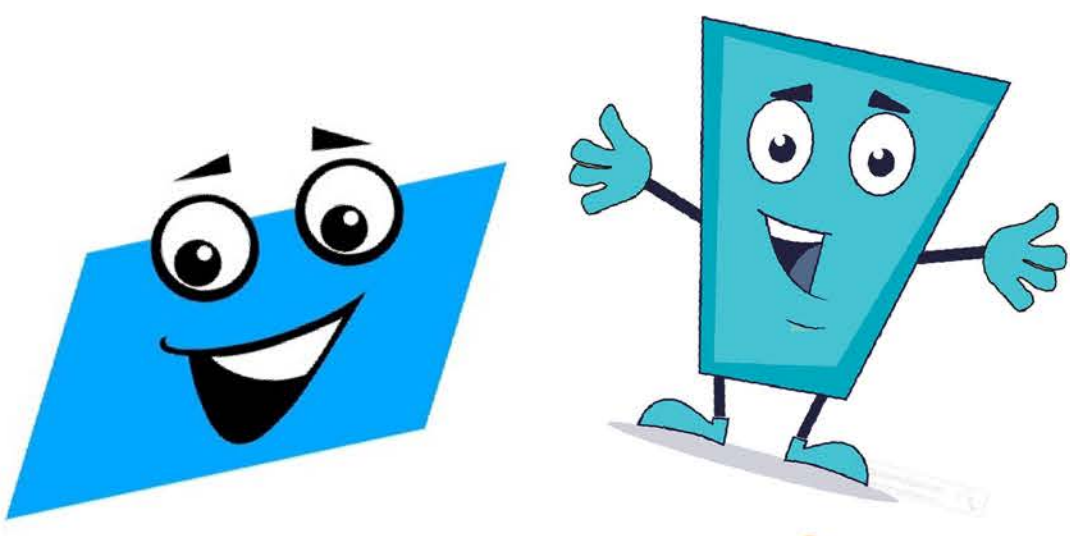
This program multiplies two natural numbers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of multiplication instead of using the programming language's built-in "*" operator to get the result directly.

The program is an upgrade of "Long Multiplication 1" (G324) with special handling of trailing zeros of the two numbers. Since the core operations of long multiplication are the same, long_multiply_core() is reused to multiply the main parts of the two numbers after separating their trailing zeros.

This program is the final version of the series of multiplication programs (G316, G324, G414).

```
Enter the first number: 5890
Enter the second number: 6500
      5 8 9 0
x       6 5 0 0
-------------------
    2 9 4 5
  3 5 3 4
-------------------
  3 8 2 8 5 0 0 0
```

Parallelograms and Trapezoids



Parallel lines

Perpendicular lines

Drawing perpendicular lines and rectangles

Distance from a point to a line

Parallelograms (base, height)

Trapezoids (top base, bottom base, legs, height)

Isosceles trapezoids and right trapezoids

Program

Counting Trapezoids

count_trapezoids.py

random, nested loop, tkinter, coordinate

The program randomly generates a specified number of line segments between two parallel lines, ensuring that none of the line segments are parallel. A trapezoid can be constructed using the two parallel lines as the top and bottom bases and any two non-intersecting line segments as the legs.

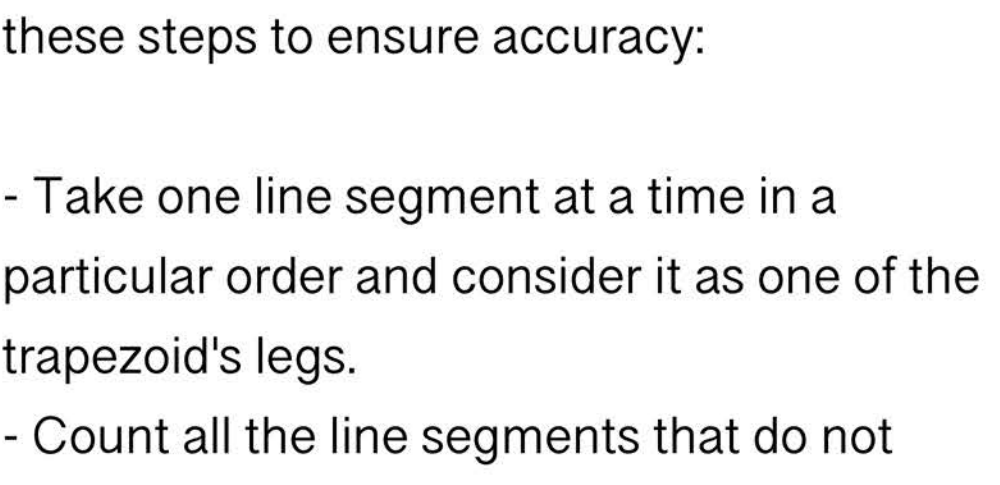
The program starts by asking the user to specify how many line segments to generate. It also asks whether these line segments can intersect. If the user doesn't confirm this, the program assumes that all line segments are non-intersecting. The user then examines the randomly generated lines to count how many trapezoids are present. After providing an answer, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown.

When counting trapezoids, you can follow these steps to ensure accuracy:

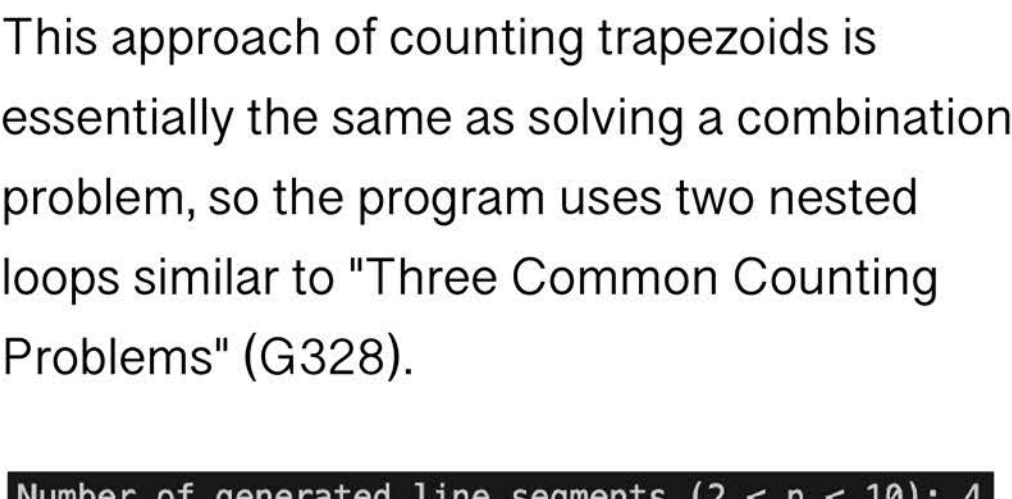
- Take one line segment at a time in a particular order and consider it as one of the trapezoid's legs.
- Count all the line segments that do not intersect with this chosen leg as the other leg - each pair of legs is a trapezoid. Look for the second leg in one direction to avoid counting the same pair of legs more than once.

This approach of counting trapezoids is essentially the same as solving a combination problem, so the program uses two nested loops similar to "Three Common Counting Problems" (G328).

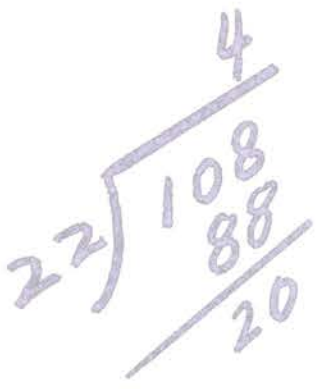
```
Number of generated line segments (2 < n < 10): 4
Are intersections allowed? (y/n) n
How many trapezoids? 6
Correct!
```

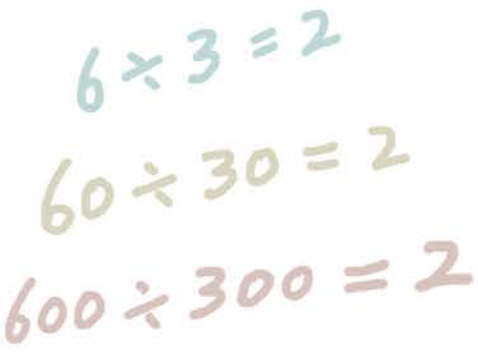



```
Number of generated line segments (2 < n < 10): 4
Are intersections allowed? (y/n) y
How many trapezoids? 3
Incorrect. The answer is 4.
```



Long Division





Mental division and estimation with 2-digit divisors

Long division with 2-digit divisors

Patterns in the quotient

Long division with dividends and divisors with trailing zeros

Program

★★★★★

Long Division

long_division.py string

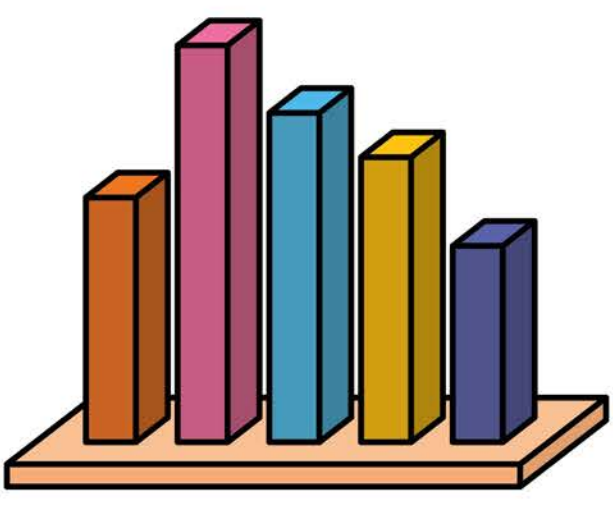
This program divides two natural numbers (the divisor must not be zero) inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of division instead of using the programming language's built-in "//" and "%" operators to get the result directly.

The basic operations of short division and long division are the same. Therefore, “Short Division” (G322) can be used for long division when the restriction of the divisor's number of digits is removed. Based on “Short Division”, this program adds the additional step to cancel the dividend and divisor's common trailing zeros. When both the dividend and divisor end in zeros, you can simplify the calculation by removing the same number of trailing zeros from both (if there's a remainder, add the same number of zeros to the end of the remainder).

This program is the final version of the series of division programs (G322, G416).

```
Enter a natrual number as the dividend: 5480
Enter a non-zero natrual number as the divisor: 360

    15
   ---
36/548
   36
   ---
   188
   180
   ---
     8

5480 / 360 = 15 ... 80
```

Bar Charts



Bar charts

Choosing appropriate scales for bar charts

Program 1

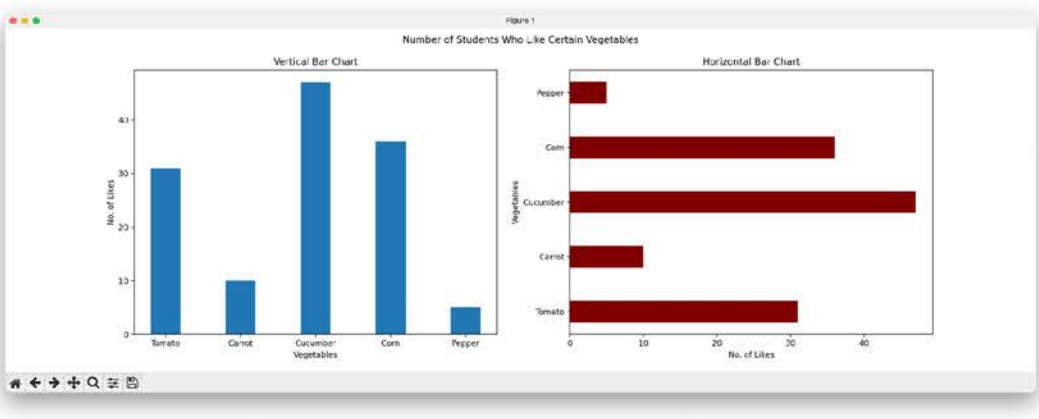
Creating Bar Charts Using Matplotlib

bar_chart.py matplotlib

This program uses the popular Python plotting library Matplotlib to create vertical and horizontal bar charts. In Matplotlib, all the plots within a single window are referred to as a Figure, and a Figure can contain one or more plots, with each plot known as an Axes. Matplotlib has two coding styles:

- Object-oriented (OO) style: Explicitly create Figures and Axes and call their methods for plotting.
- Pyplot style: Implicitly create and manage Figures and Axes using pyplot and use pyplot's functions for plotting.

Matplotlib generally suggests using the object-oriented style, particularly for complicated plots. However, pyplot can be very convenient for quick interactive work. For comparison, this program uses Matplotlib to draw bar charts in two Figures, using one coding style for each. Each Figure contains two Axes arranged horizontally: one for the vertical bar chart and the other for the horizontal bar chart.



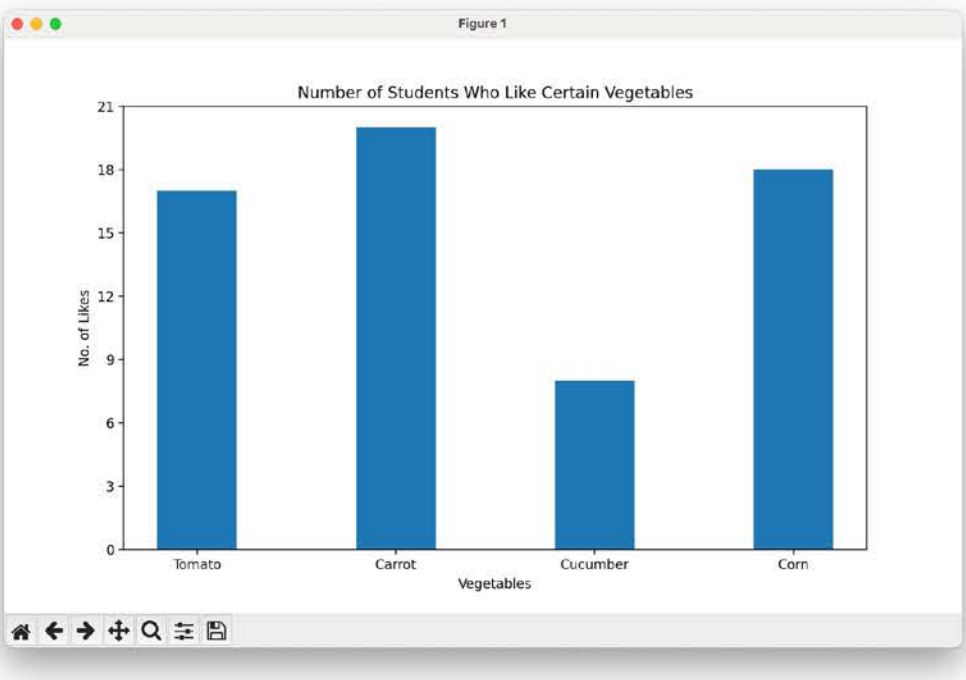
Program 2

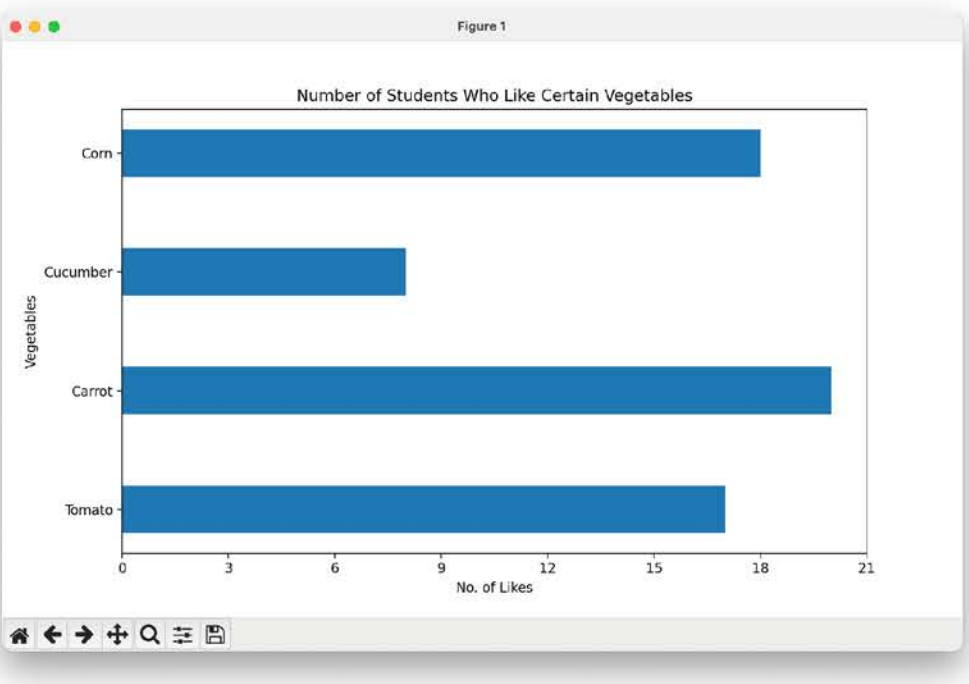
Creating Subclass of Table Class to Draw Bar Charts

data1.py matplotlib, class

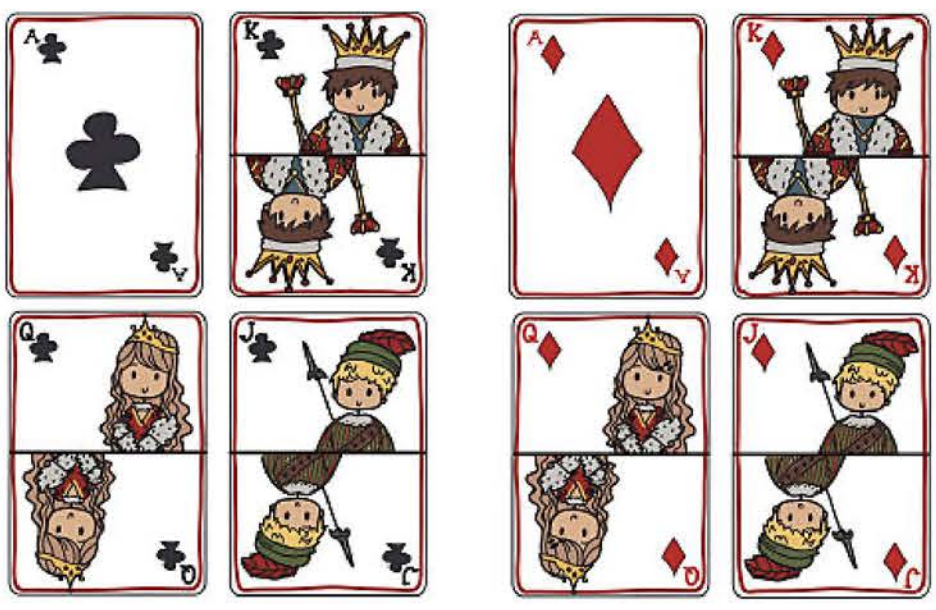
This program is based on the previous program, "Creating and Displaying Tables" (G323), and introduces a subclass called Data. The Data class inherits the properties and methods of the Table class and adds two new methods, bar() and barh(), for drawing vertical and horizontal bar charts. This means that the data in the Data class can be displayed as a table in the command line interface and as bar charts in a graphical interface. In future programs, more methods for drawing grouped bar charts and line charts will be added to the Data class.

	Tomato	Carrot	Cucumber	Corn
No. of Likes	17	20	8	18





Optimization



Overall planning and optimization

Winning strategy

Program

Counting Game

counting_game.py

random

Here are the rules of the Counting Game: Two players take turns counting. Each turn, they choose a number within a given range and add it to the shared counter. Whoever makes the counter reach the target number will win. This program simulates this game with a player and computer. The player counts first.

There is a winning strategy to this game. Take the addition range 1 to 3 and the target number 21 - to make the counter reach 21, you first need to make the counter 17 because no matter if the opponent chooses +1, +2, or +3, you can add to 21 on your next turn. Similarly, if you want to make the counter 17, you have to make the counter 13, and so on. Using this method, you will have a list of "winning numbers:" (21, 17, 13, 9, 5, 1). So, as long as 0 is not a winning number, the player who counts first can count the first winning number, leading to guaranteed victory. The program also uses this strategy. So, If the player makes a mistake, the computer will make the counter a winning number and eventually win the game.

```
Each turn, you can choose a number from [1, 2] to add to the shared counter.
Whoever makes the counter reach exactly 10 wins.

Current counter: 0 -> Target number: 10
Your turn: 1

Current counter: 1 -> Target number: 10
Computer's turn: 2

Current counter: 3 -> Target number: 10
Your turn: 2

Current counter: 5 -> Target number: 10
Computer's turn: 2

Current counter: 7 -> Target number: 10
Your turn: 1

Current counter: 8 -> Target number: 10
Computer's turn: 2

Current counter: 10 = Target number: 10
Computer wins!
```

Order of Operations

$96 \div [(12+4) \times 2]$

$= 96 \div [16 \times 2]$

$= 96 \div 32$

$= 3$

Addition

Sum = Addend + Addend

Addend = Sum - Other Addend

Subtraction

Difference = Minuend - Subtrahend

Subtrahend = Minuend - Difference

Minuend = Subtrahend + Difference

Multiplication

Product = Factor × Factor

Factor = Product ÷ Other Factor

Division

Quotient = Dividend ÷ Divisor

Divisor = Dividend ÷ Quotient

Dividend = Quotient × Divisor

Mixed Arithmetic Operations, Parentheses ( ), [ ], { }

Order of Operations in Mixed Operations with Parentheses

Program

Evaluate Arithmetic Expressions

eval_expressions.py string

The program takes user input of any arithmetic expression in text form and displays the calculation result on the screen.

The program can parse arithmetic expressions containing addition, subtraction, multiplication, division, and parentheses. The algorithm used simulates the steps people take when performing calculations:

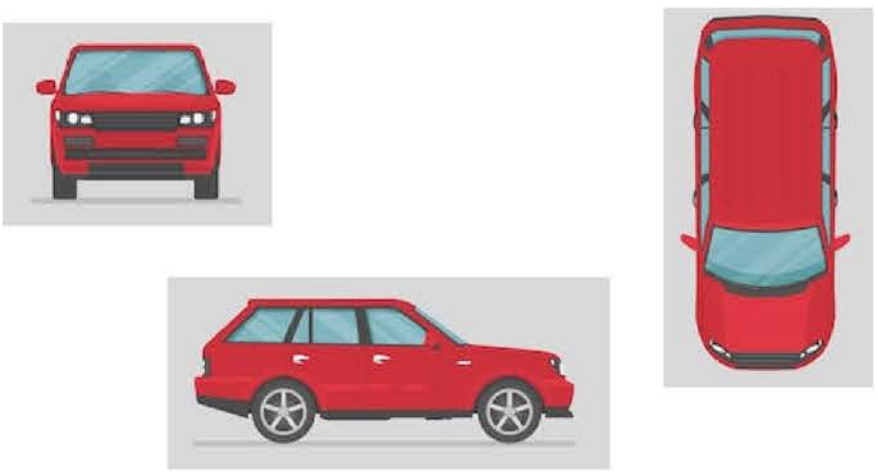
1. Scan the expression string to get operands and operators, and set the precedence level for each operator according to the order of operations learned in this unit.
2. Then, in order of precedence, and from left to right in cases of equal precedence, calculate each part until the final result is obtained.

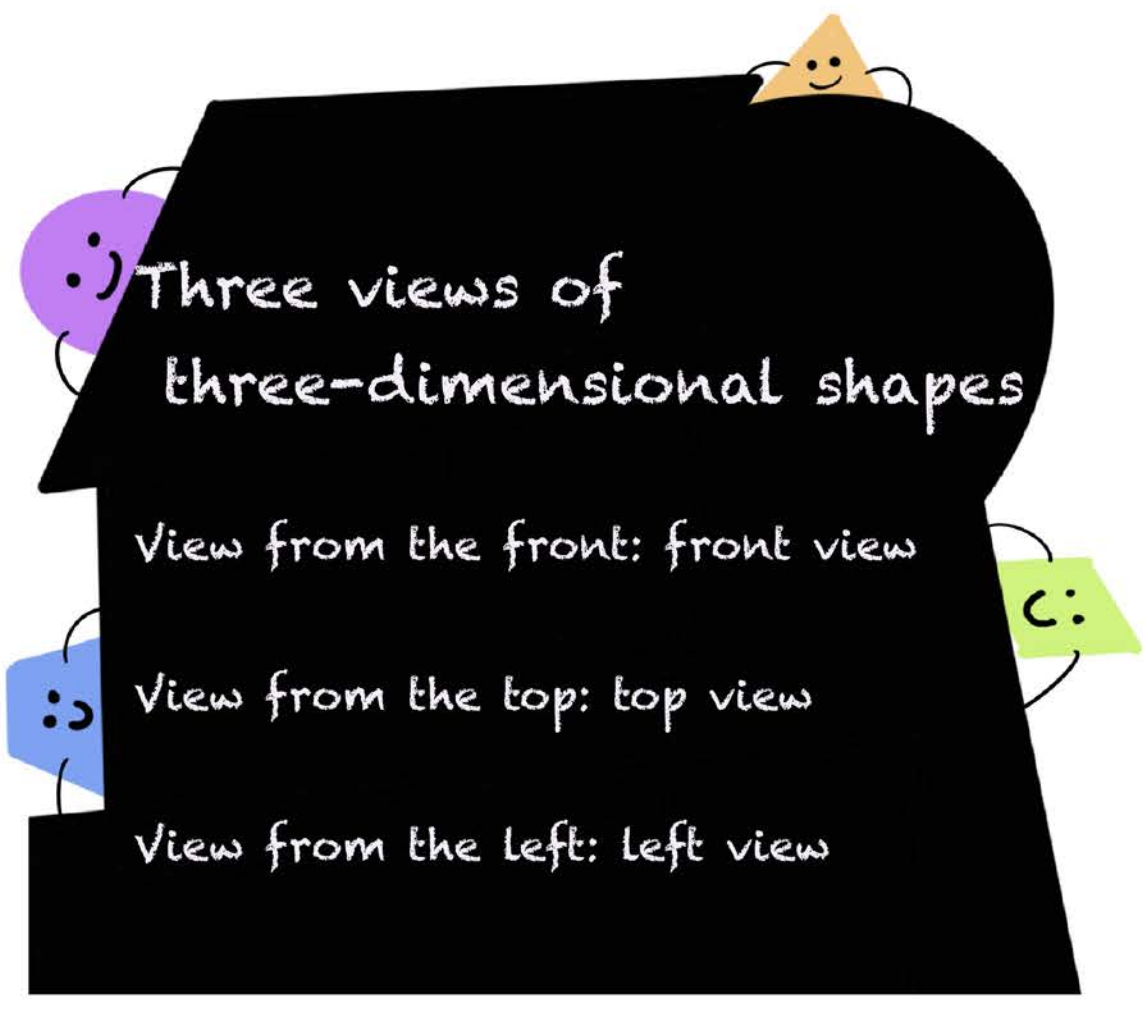
The program doesn't use stack or recursion algorithms for evaluating expressions. Instead, it simulates our actual calculation process, which is less efficient but more understandable.

Characters in the expression other than digits and '+-*/()' are ignored. Nested parentheses all use '().'

```
Enter an arithmetic expression:
96/((12+4)*2)
3

Enter an arithmetic expression:
96 / ((1 2+ 4)*2 )
3
```

Observing Objects





Program

Three Views of Cubes

cubes_3view.py

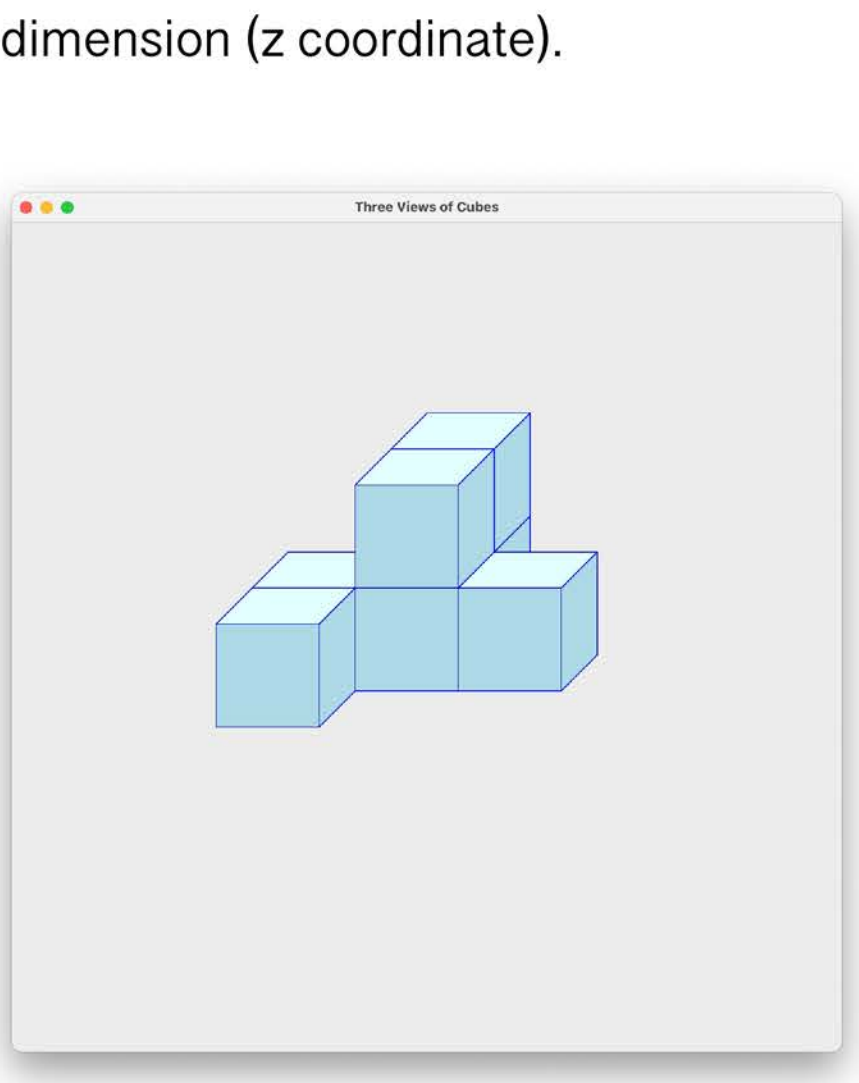
tkinter, coordinate, 2d list, random, class

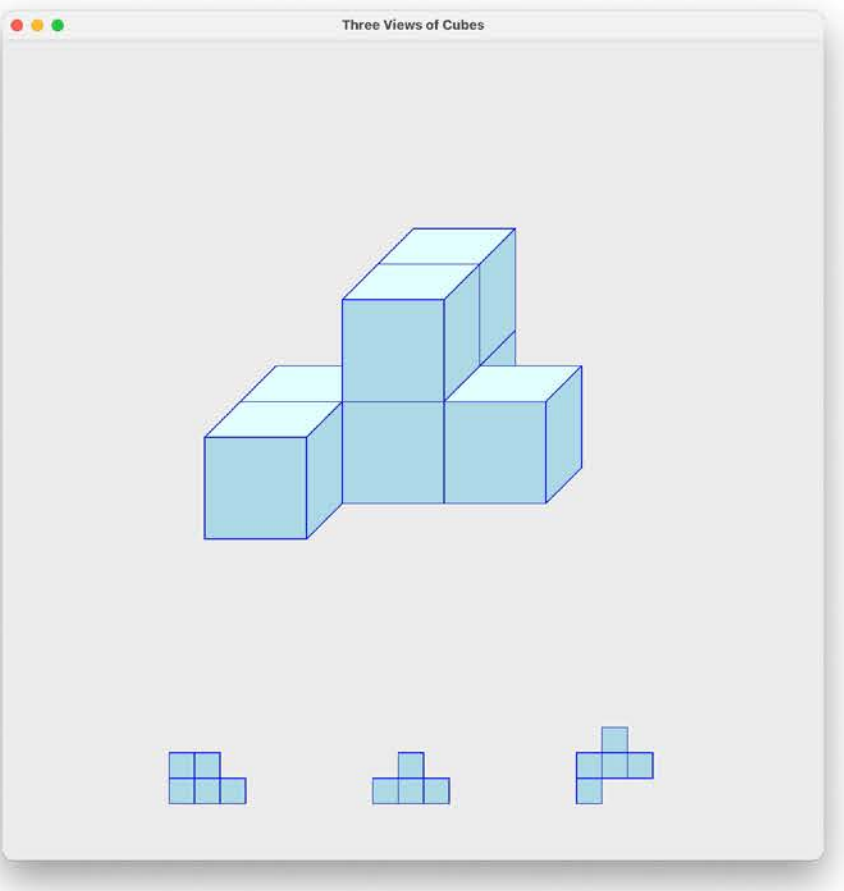
The program randomly generates and draws an object made of cubes. Press the spacebar to display the three views of the object at the bottom of the window. From left to right, they are the left view, front view, and top view. Press the spacebar again to generate a new object.

The spatial position (column, row, layer) of each cube is stored in a list, and all cubes are contained in the object. So, the object is represented by a two-dimensional list.

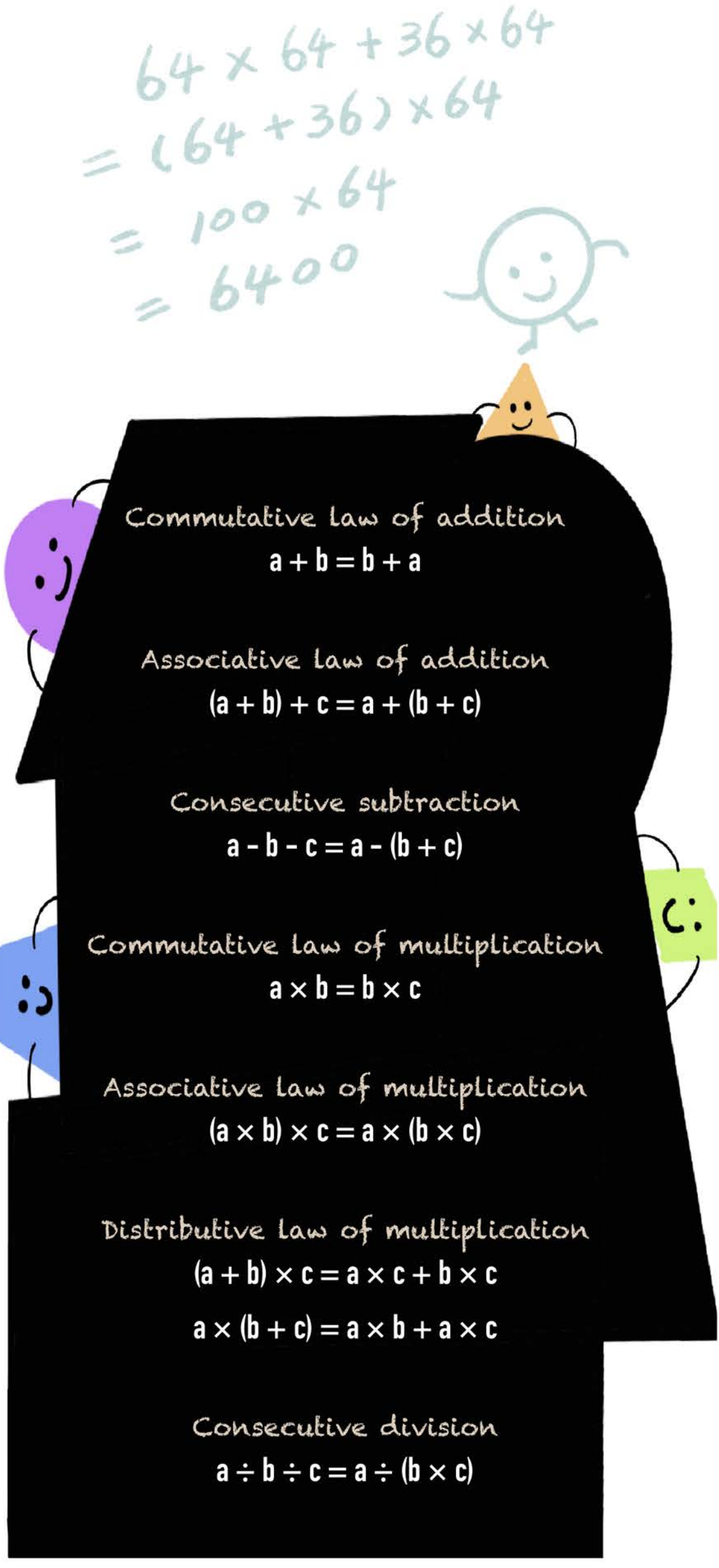
What is drawn on the canvas will cover the previous drawings in the same position. So, the cubes only need to be drawn from left to right, back to front, and bottom to top to achieve an occlusion effect.

The object is flattened to a view by ignoring one of the three dimensions of the object. For example, a top view can be drawn by ignoring the layer dimension (z coordinate).





Basic Laws of Operation



Program

Solve 24

solve_24.py itertools, nested loops, dictionary, algorithm

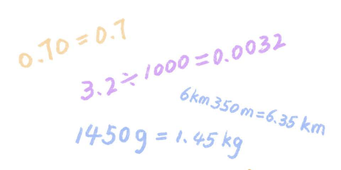
The 24 Game is played with a deck of playing cards with all the face cards removed. Randomly draw four cards, and the first player that uses all values on the cards, elementary arithmetic operations (+, -, *, /), and parentheses to come up with 24 wins. There are also some variants of this game, such as using J, Q, and K or allowing more operations.

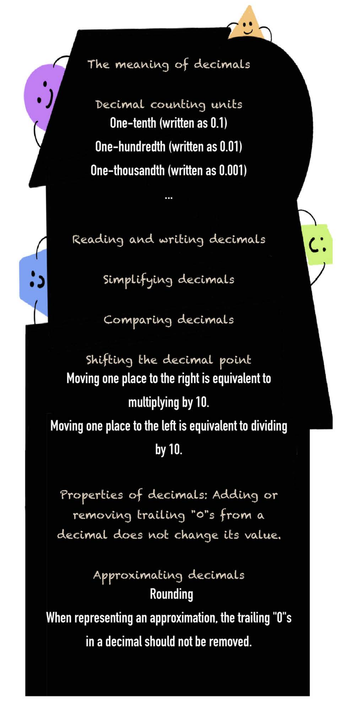
Given four numbers, the program will display all solutions that are not equivalent. Here are some specific explanations:

- The program uses brute force to try every arithmetic expression and uses the calc() function in eval_expressions.py (G421) to calculate the result of the expressions.
- Many solutions, such as ones using commutative or associative laws, are equivalent. To check whether two expressions are equivalent, the four given numbers are mapped onto another four random numbers, and these new numbers replace the operands in the two expressions. If the results of the two expressions are still the same, they are equivalent.
- When displaying solutions, parentheses are only used when they are required.
- The programs can be used to solve problems similar to the 24 Game. In fact, users can enter any amount of numbers and any target number (not necessarily 24) and get all non-equivalent solutions.

```
>>> solve(7, 8, 5, 4)

8 + 7 + 5 + 4
(8 - 7 + 5) × 4
8 × (7 + 5) / 4
(8 + 4) × (7 - 5)

>>> solve(7, 8, 5, 4, target=40)

(8 + 7 - 5) × 4
(7 + 5) × 4 - 8

>>>
```

Meaning and Properties of Decimals





Program

Decimal Practice 2

decimal_practice2.py random, decimal

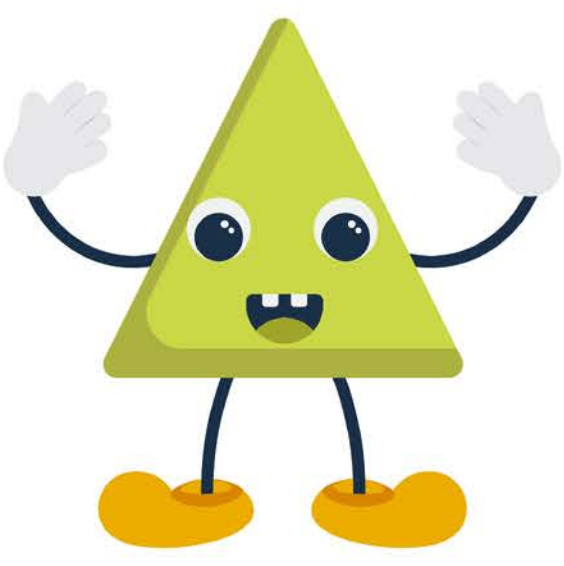
Randomly generate a specified number of decimal practice questions. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set; the default is 100 points). There are three types of questions - users can practice one type at a time or mix questions from multiple types. The three question types are as follows:

1. Approximating Decimals: The program presents a random decimal number and generates instructions on how many decimal places to retain or up to which place to be precise.
2. Decimal Point Shifting Exercises: The program presents a random decimal number, and you will practice moving the decimal point to the right or left by calculating the product or quotient when multiplying or dividing the decimal by 10, 100, or 1000.
3. Two-way unit conversion from "Decimal Practice 1" (G327). For unit conversions that involve shifting the decimal point significantly to the left, third-grade students may have used fractions for answers. This unit provides an opportunity to practice using decimals for answers. The answers provided by the program may sometimes appear as "1e-06" (equivalent to "1e-6"), which is scientific notation representing moving the decimal point of 1.0 to the left by 6 places, i.e., 0.000001. Similarly, "1e6" is moving the decimal point of 1.0 to the right by 6 places, i.e., 1000000.

The default data type for representing decimals in Python is float. Float operation is very efficient, but there are often errors, e.g., 1.1 + 2.2 = 3.3000000000000003. There is a data type, Decimal, in the decimal module of the Python standard library that is specially used to represent decimals. Although the efficiency is not as good as float, it can obtain accurate results of decimal operations. The Decimal data type is used in the program to calculate correct answers for questions of rounding decimals and moving decimal points.

```
Question 1/4:  19.971 rounded to the nearest tenth
? 20
Incorrect. The answer is 20.0.

Question 2/4:  84.68 / 1000 =
? 0.08468
Well done!

Question 3/4:  98.4758 rounded to 2 decimal places
? 98.48
Excellent!

Question 4/4:  1g = __t
? 0.000001
Perfect!

Your score is 75/100.
```

Triangles



Three sides, angles, vertices

Height and base of a triangle

Stability of triangles

The shortest line segment among all lines between two points is the distance between those two points.

The sum of any two sides of a triangle is greater than the third side.

Classification of triangles

Based on angles: acute triangle, obtuse triangle, right triangle.

Based on sides: isosceles triangle, equilateral triangle.

The pattern for the sum of interior angles of polygons

The sum of the interior angles of a triangle is 180°.

The sum of the interior angles of a quadrilateral is 360°.

...

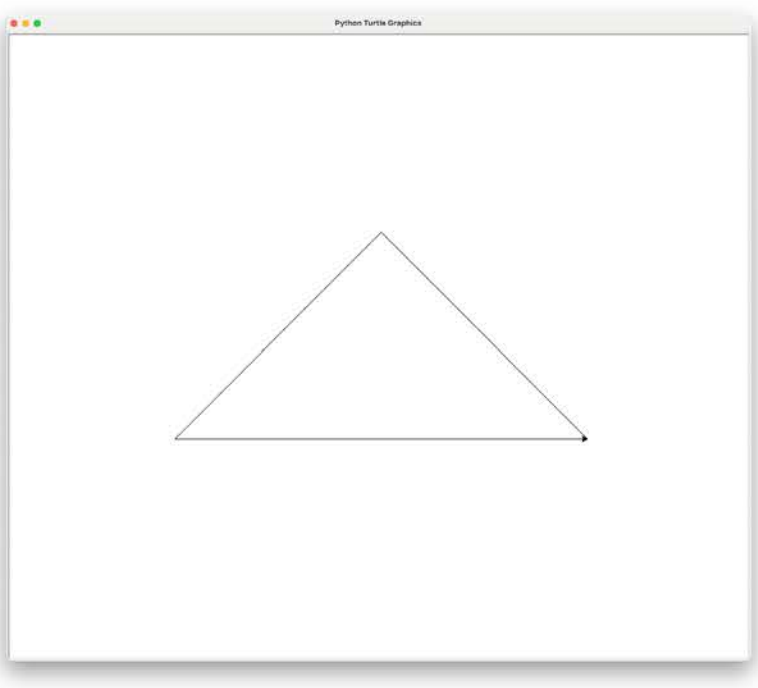
Program 1

Draw Isosceles Triangles

isosceles_triangle.py turtle

Given the vertex angle and length of the legs, the program draws the isosceles triangle and displays its base angle and base length. Without using trigonometric functions, the length of the base is obtained by using turtle's distance() method to return the distance between its two endpoints.

```
Enter the vertex angle: 90
Enter the length of legs (0 - 500): 500

Vertex Angle: 90  Base Angle: 45  Legs: 500  Base: 707
```



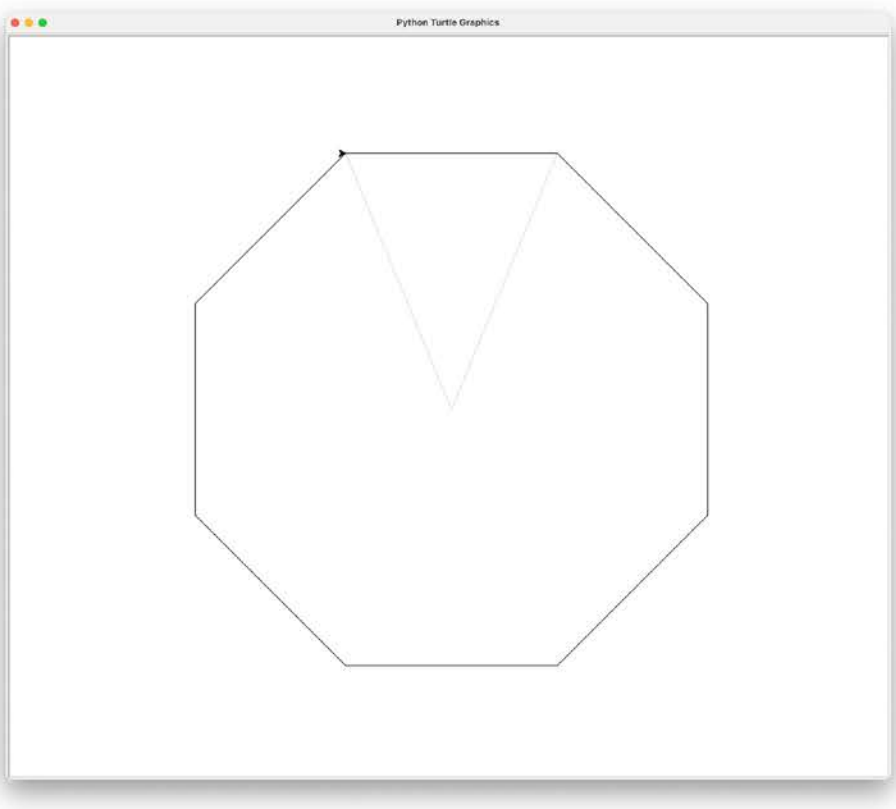
Program 2

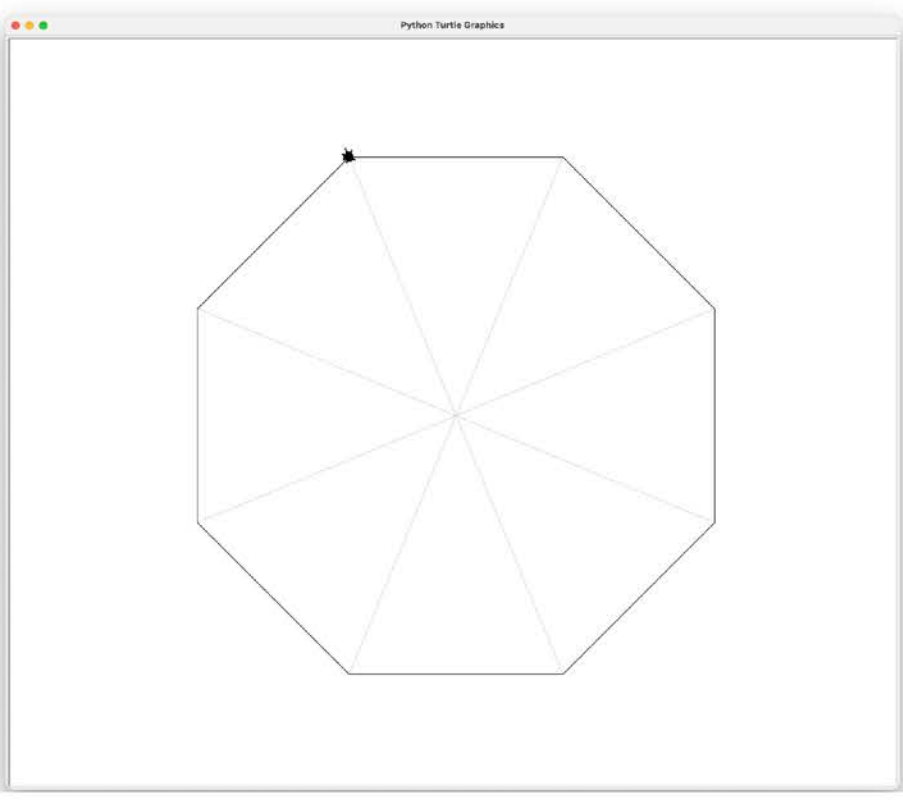
Draw Regular Polygons

regular_polygon.py turtle

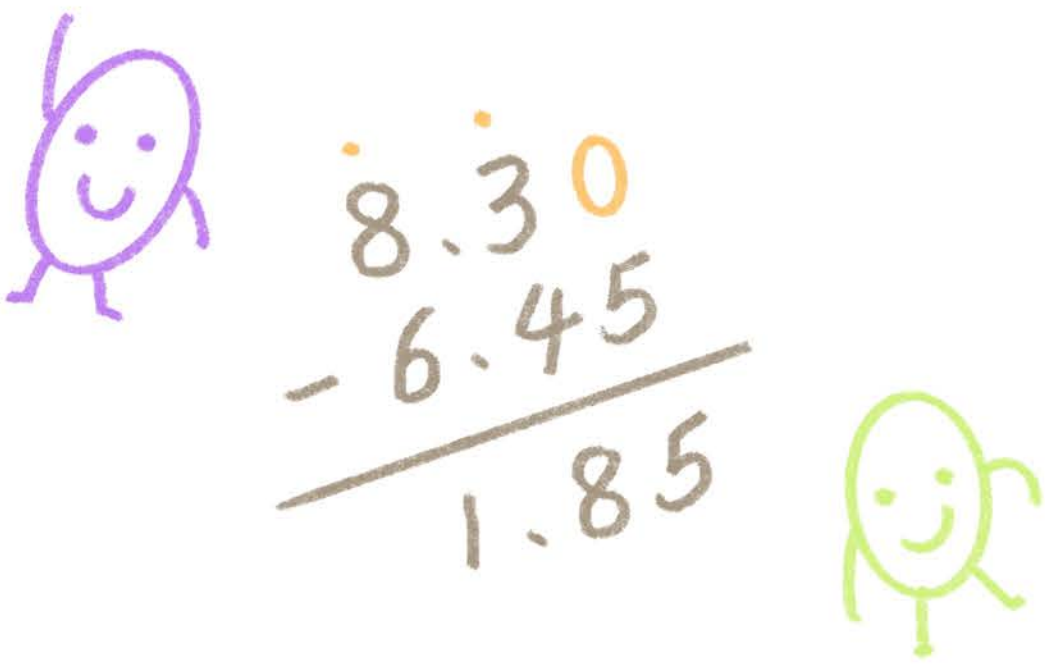
This unit extends the study from triangles to polygons. The second program is about drawing regular polygons (given the distance from the center to a vertex). The program uses two methods:

1. As long as you know the side lengths, you can use turtle to go around and draw a regular polygon. First, find the two adjacent vertices of the regular polygon, and then use turtle's distance() method to return their distance, which is the side length of the regular polygon.
2. Use two turtles. The first turtle finds vertices in order, while the second turtle follows from the previous vertex and draws an edge. When the first turtle goes around and finds all the vertices, the second turtle completes the regular polygon.





Addition and Subtraction of Decimals



Decimal addition and subtraction in vertical form (aligning the decimal points)

Simplifying calculations using laws of addition

Program

Addition and Subtraction of Decimals in Vertical Form

add_sub_decimals.py string

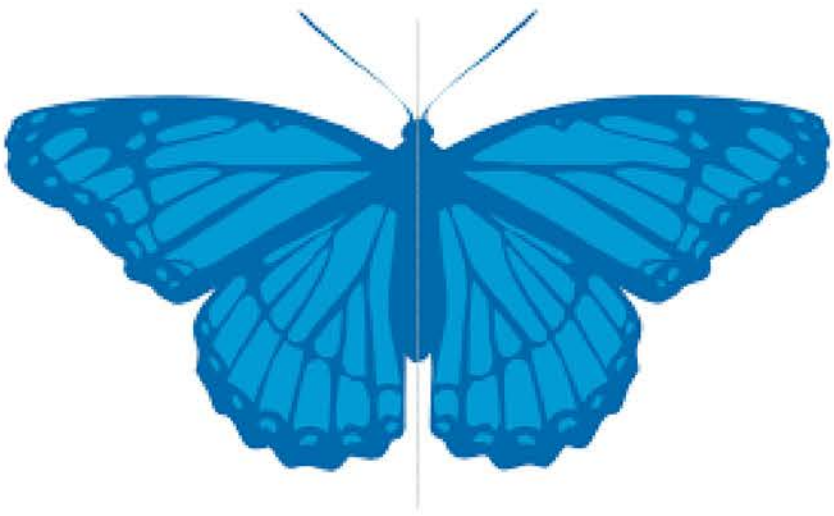
This program sums or subtracts two decimals or integers inputted by the user and displays the result on the screen in vertical form. The program realistically simulates the vertical operation process of addition instead of using the programming language's built-in "+" operator to get the result directly.

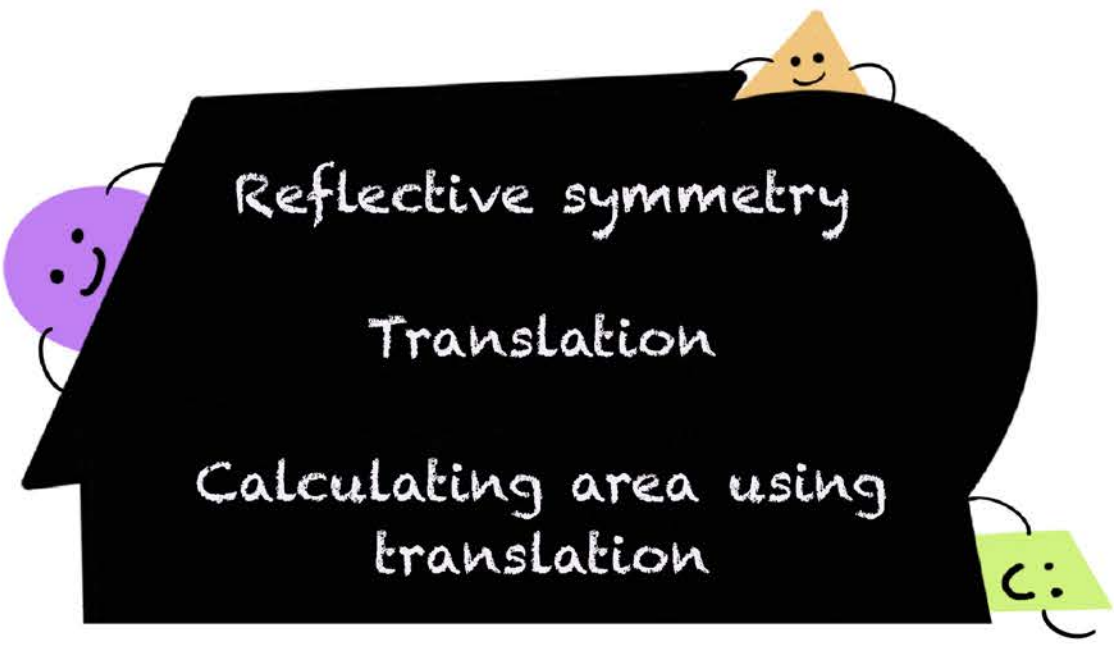
The addition and subtraction of decimals with the decimal points aligned are almost the same as the addition and subtraction of integers. So, the program simulates the calculation process of decimals based on reorganized elements of “Vertical Addition” (G314) and “Vertical Subtraction” (G314).

1. Add zeros to the end of the decimal with fewer decimal places to make both decimals have the same number of decimal places - making their decimal points aligned.
2. Remove decimal points of operands and do integer addition or subtraction (same as in the programs of G314).
3. Add a decimal point to the result at the same position as operands' decimal points.
4. Remove operands' trailing zeros added at step 1, except those of the minuend.
5. Display the vertical columns (same as in the programs of G314).
6. Remove trailing zeros of the result and display the simplified result.

```
Enter the first decimal: 23.5
Enter the second decimal: 4.85
Choose between addition and subtraction (+ or -)? +

  2 3 . 5
+   4 . 8 5
-----------
  2 8 . 3 5

The simplified result is: 28.35


Enter the first decimal: 1
Enter the second decimal: 0.01
Choose between addition and subtraction (+ or -)? -

  1 . 0 0
- 0 . 0 1
---------
  0 . 9 9

The simplified result is: 0.99


Enter the first decimal: 0.99
Enter the second decimal: 0.01
Choose between addition and subtraction (+ or -)? +

  0 . 9 9
+ 0 . 0 1
---------
  1 . 0 0

The simplified result is: 1
```

Reflective Symmetry





Program

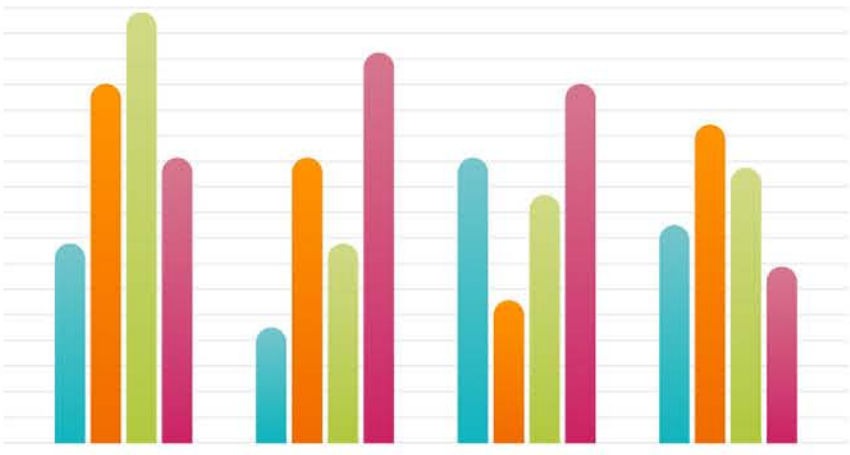
Generate Reflective Symmetric Shapes

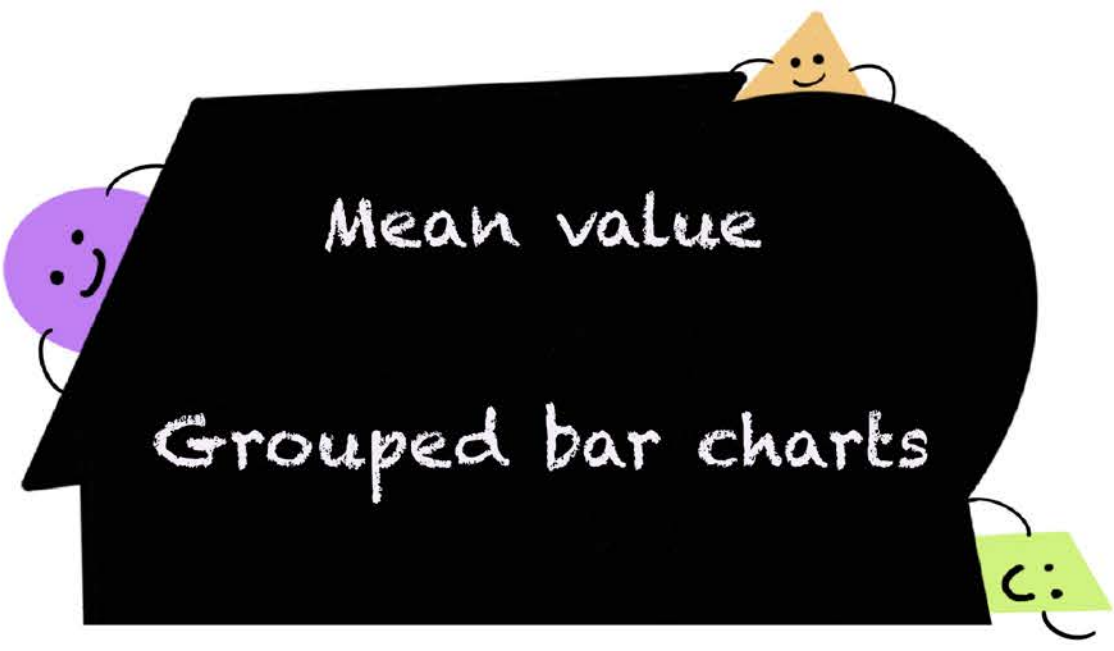
reflective_symmetry.py random, turtle

The program randomly generates a left-right reflective symmetric shape with the axis of symmetry placed in the middle of the window. On one side of the axis of symmetry, the program generates a random number of points based on user input and mirrors these points to the other side of the axis of symmetry. Connecting all these points forms a reflective symmetric shape.

```
Generate how many points on one side (1-50)? 20
```

Mean Value and Grouped Bar Charts





Program

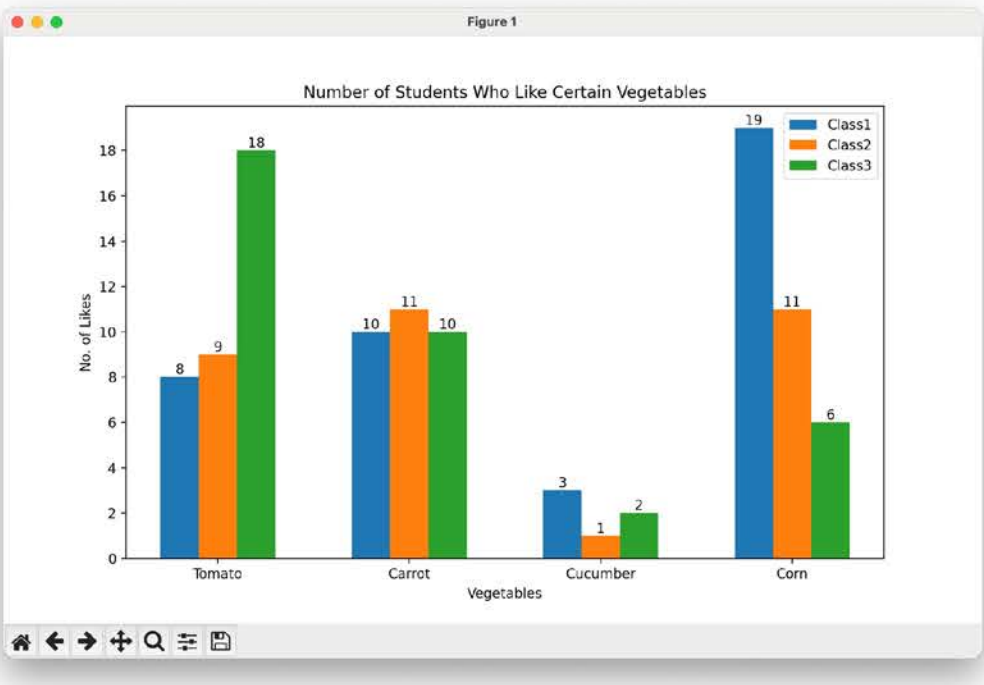
Mean Value and Grouped Bar Charts

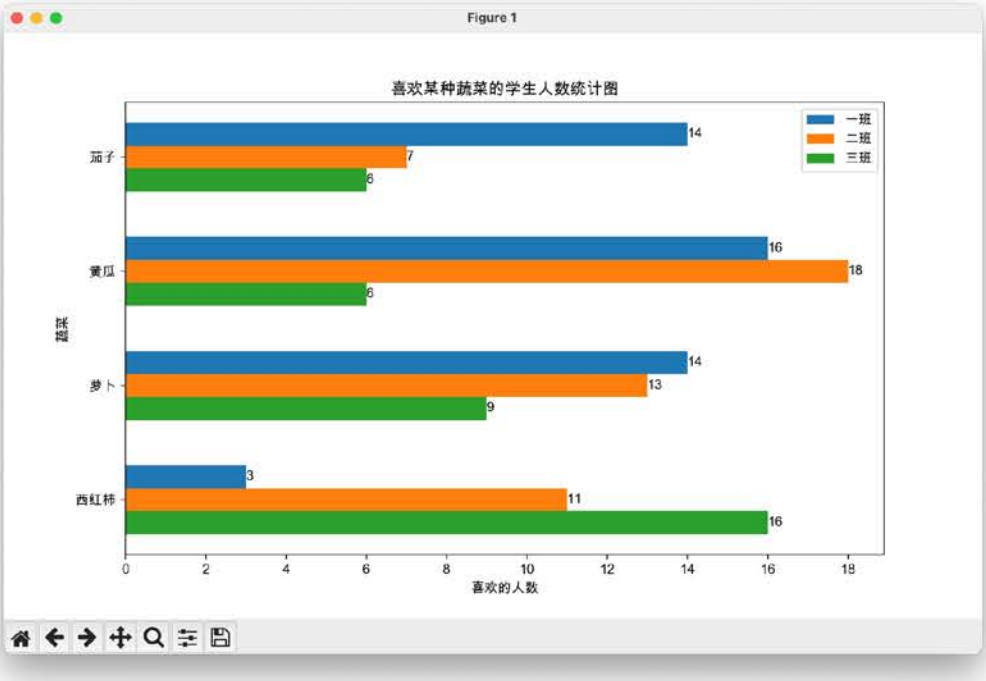
data2.py matplotlib, class, 2d list

The program improves the Data class in “Creating Subclass of Table Class to Draw Bar Charts ” (G417). bar() and barh() methods of the Data class can now draw vertical and horizontal grouped bar charts using data from more than one row. The methods avg_row() and avg_col() are also added to the Data class. They calculate the average (mean value) of a given row or column.

	Tomato	Carrot	Cucumber	Corn
Class1	8	10	3	19
Class2	9	11	1	11
Class3	18	10	2	6

```
The average of each row is: [10, 8, 9]
The average of each column is: [12, 10, 2, 12]
```





Chicken and Rabbit Problem

Solving the chicken and rabbit problem

List method

Assumption method

Lifting legs method

Program

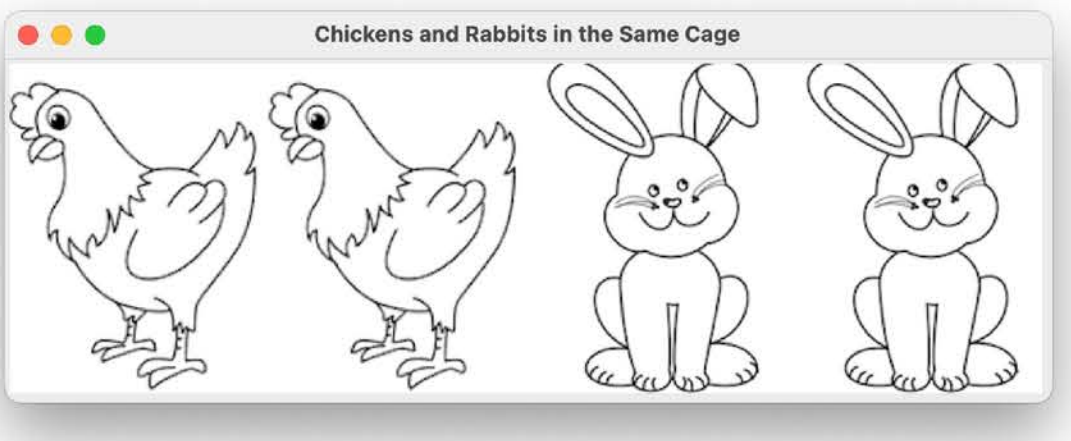
Chicken and Rabbit Problem

chicken_rabbit.py tkinter, random

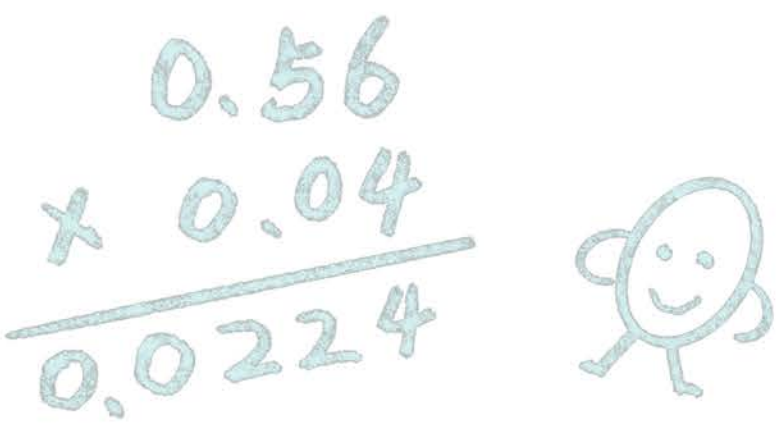
The “Chicken and Rabbit Problem” is a famous ancient Chinese math problem from around 1,500 years ago. It involves a cage with a certain number of chickens and rabbits. By counting the number of heads and the number of legs, you need to determine how many chickens and rabbits are in the cage (rabbits have four legs, whereas chickens have two).

This program generates a random variation of the problem. After the user inputs an answer, it provides feedback on whether it is correct or incorrect and displays the correct answer. The program also displays images of chickens and rabbits in the window, showing the correct number of each, providing an intuitive way to understand the problem.

```
There are 4 heads and 12 legs.
How many chickens? 2
How many rabbits? 2
Correct! There are 2 chickens and 2 rabbits.
```



Decimal Multiplication



Decimal long multiplication

Product approximation

Simplifying calculation using the laws of multiplication

Estimation

Program

Long Multiplication of Decimals

multiply_decimals.py string

This program multiplies two decimals (or integers) inputted by the user and displays the result. The program realistically simulates the vertical operation process of multiplication instead of using the programming language's built-in "*" operator to get the result directly. The process of decimal multiplication is as follows:

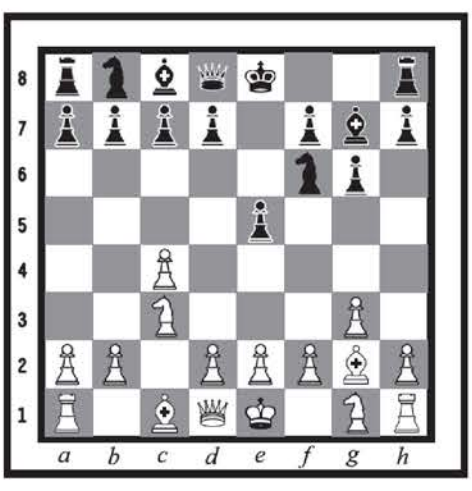
1. Ignore the decimal points of the two factors and calculate the product of corresponding integers by long multiplication.
2. Add a decimal point to the product according to the factors' number of decimal places.
3. If the product doesn't have enough digits for decimal places, add leading zeros.
4. Remove trailing zeros of the product if any exist.

When implementing the first step of integer multiplication in vertical format, the program calls the long_multiply_core() function from "Long Multiplication 1" (G324) to obtain the calculation result.

The focus of this unit is on understanding and mastering the process of decimal multiplication. Since displaying decimal multiplication in vertical format involves various considerations at the display level, the program only provides the result of decimal multiplication. Interested learners can challenge themselves to display the vertical form of decimal multiplication.

```
Enter the first decimal: 5.5
Enter the second decimal: 6
5.5 × 6 = 33

Enter the first decimal: 0.55
Enter the second decimal: 0.6
0.55 × 0.6 = 0.33
```

Position



Representing position using coordinates

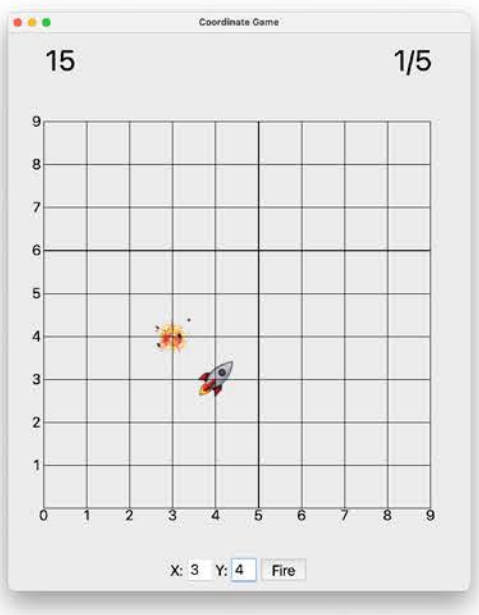
Program 1

Input Coordinates Based on Positions

input_coordinate.py tkinter, random

In this game, a rocket is randomly placed in a position, and the player's task is to input the coordinates of the rocket's location and launch a missile to shoot it down. If the specified number of rockets is shot down within the set time limit, the mission is successful. The time limit and the number of rockets can be adjusted within the program.

You can use the mouse or TAB key to change focus when inputting and submitting coordinates, but the fastest way is pressing RETURN in entry_x to change focus to entry_y and pressing RETURN again to submit.

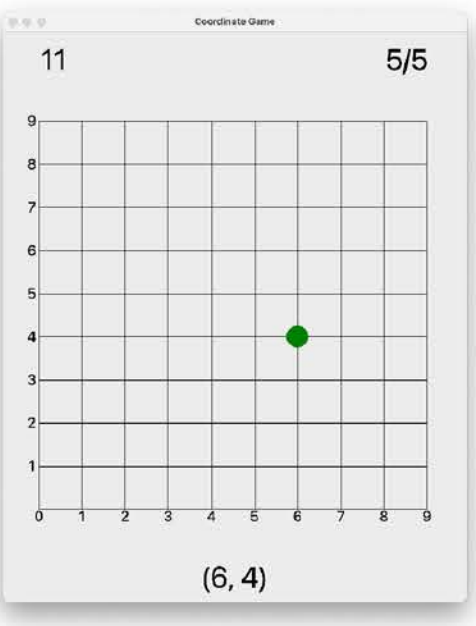


Program 2

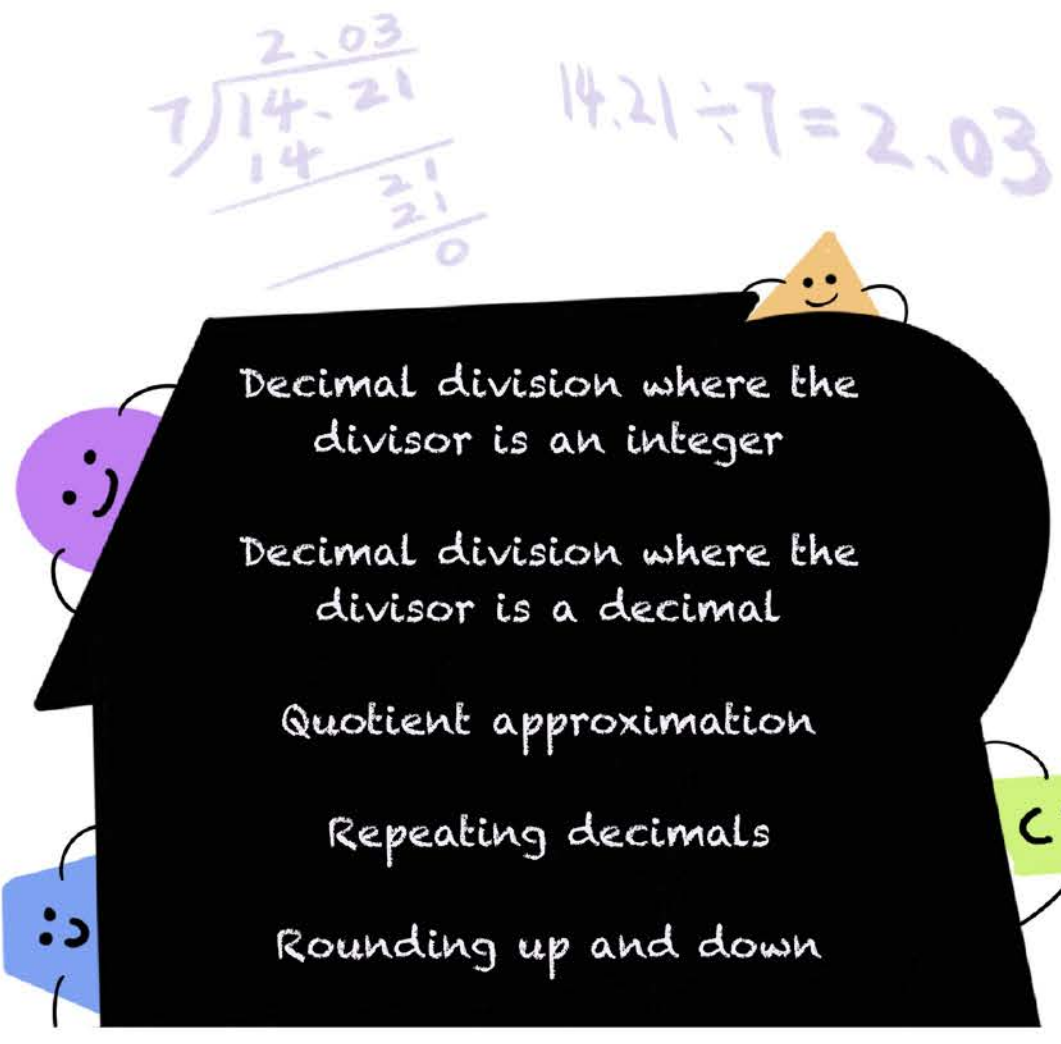
Click on Positions Based on Coordinates

click_position.py tkinter, random

In this game, a specified number of coordinates are generated randomly, and the player needs to click on the corresponding positions with the mouse. If the correct position is clicked, a green circle is shown; if the wrong position is clicked, a red circle is shown. If the player correctly identifies all coordinates within the set time limit, the mission is successful. The time limit and the number of coordinates generated can be adjusted within the program.



Decimal Division



Program 1

Long Division of Decimals

divide_decimals.py string, decimal

The program will, using long division, calculate the quotient of two decimals (or integers) entered by the user. If the quotient is a finite decimal, the program divides until there is no remainder, providing the exact value of the quotient. If the quotient is an infinitely repeating decimal, the program continues to divide until it finds the repeating pattern and represents the quotient in a format that contains the repeating part within parentheses. The program realistically simulates the vertical operation process of division instead of using the programming language's built-in "/" operator to get the result directly.

The steps in decimal division are as follows:

1. Convert the divisor to an integer and adjust the decimal point position of the dividend accordingly.
2. Do long division with the quotient's decimal point aligned with the dividend's.
3. The long division process won't end until the quotient of a finite decimal or the repeating part of a repeating decimal is obtained. Zeros will be added to the end of the dividend if necessary.

The focus of this unit is on understanding and mastering the process of decimal division. Since displaying decimal division in vertical format involves various considerations at the display level, the program only provides the result of decimal division. Interested learners can challenge themselves to display the vertical form of decimal division.

```
Enter the dividend: 7.86
Enter the non-zero divisor: 1.3
The quotient is: 6.0(461538)
```

Program 2

Practice Converting Common Fractions to Decimals

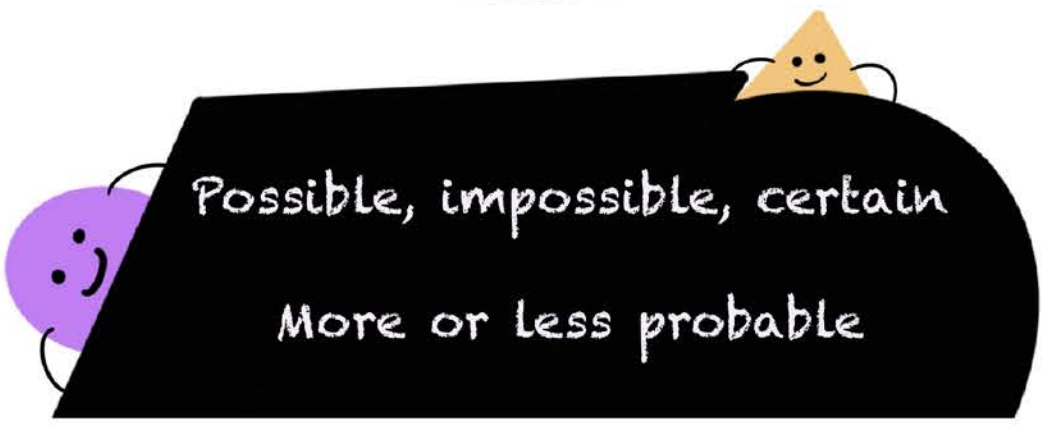
fraction_to_decimal.py random

Randomly generate a specified number of practice questions for converting common fractions with denominators less than to decimals. Every time a question is answered, the program will display whether the answer is correct or incorrect. The correct answer will be given in the case of an incorrect answer. The total score will be displayed after all questions are completed. (The full score value can be set, the default is 100 points).

Converting fractions to decimals is as simple as dividing the numerator by the denominator. The main goal of these exercises is to help learners become familiar with the decimal equivalents of common fractions. For fractions equating to repeating decimals, the decimals are rounded to three decimal places.

```
Convert Fractions to Decimals.
(For repeating decimals, round to 3 decimal places.)

Question 1/4:  3/4
Decimal? 0.75
Splendid!

Question 2/4:  1/5
Decimal? 0.5
Incorrect. The answer is 0.2.

Question 3/4:  2/9
Decimal? 0.222
Super!

Question 4/4:  1/6
Decimal? 0.167
Wonderful!
```

Probability



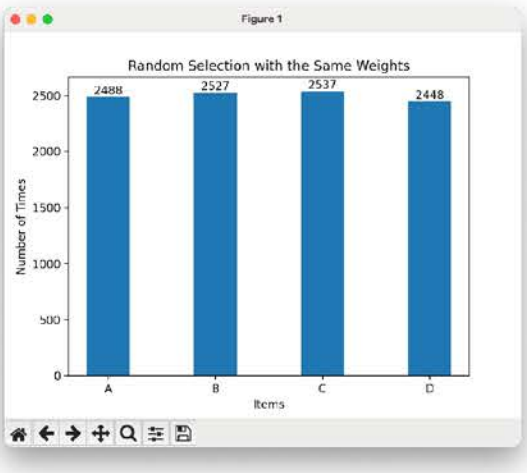
Program 1

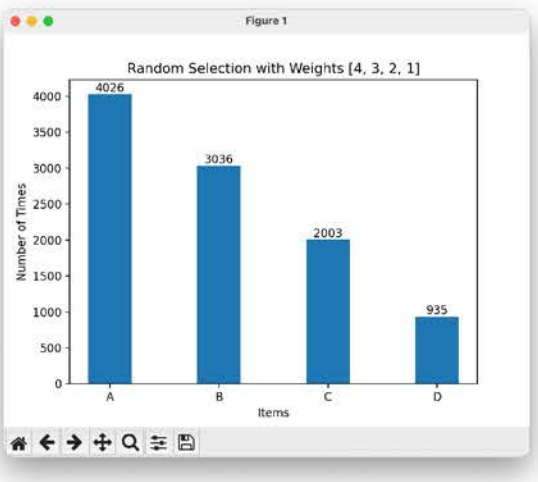
Random Selection with Weights

random_with_weights.py random, dictionary, matplotlib

When choosing within a given range using random functions such as random(), randint(), choice(), etc., all options are equally likely. But what if we want options to have different likelihoods of being selected? The program uses 4 methods to implement random selection with different weights. Items with greater weights are more likely to be selected.

The program conducts 10,000 random trials for each method, stores the statistical data in a dictionary, and visualizes the results as a bar chart using Matplotlib. The bar chart illustrates that the number of times each option is randomly selected aligns with its probability (weight).



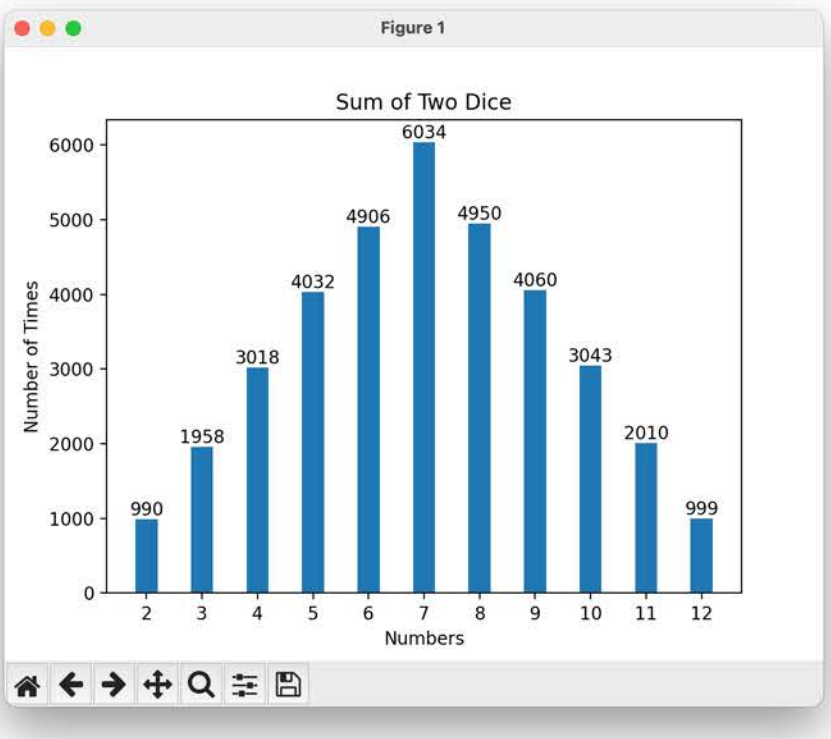


Program 2

Sum of Two Dice Rolls

two_dice.py random, dictionary, matplotlib

The sum of two dice rolls can be any value between 2 and 12, but the likelihood of each outcome is different. The program simulates rolling two dice 36,000 times, stores the statistical data in a dictionary, and creates a bar chart using Matplotlib to visualize the different possible sums. It can be seen from the bar chart that 7 is the most likely outcome, occurring about 6,000 times. Conversely, 2 and 12 are the least likely, occurring about 1,000 times each.



Simple Equations

$$8 \cdot (5x - 12) = 24$$

Representing numbers with letters

Expressing laws and formulas with letters

Expressions containing letters

Equations containing unknowns

Properties of equations

1. Adding or subtracting the same number on both sides of an equation keeps both sides equal.
2. Multiplying or dividing both sides of an equation by the same non-zero number keeps both sides equal.

Solving equations

Solving equations for real-world problems

Program

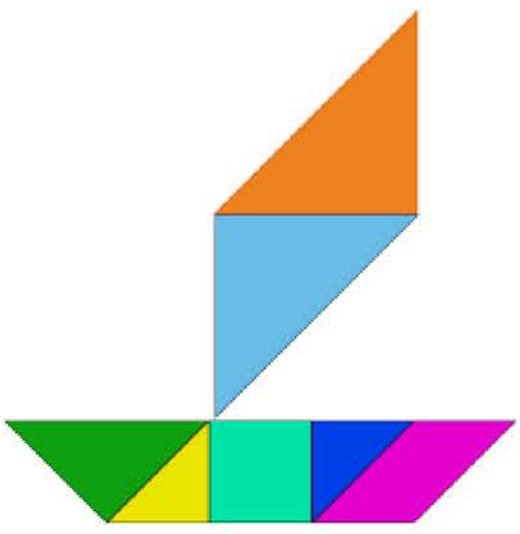
Solving Chicken and Rabbit Problem Using Equations

chicken_rabbit_equation.py string

This program guides users through all the steps of solving an applied problem using equations. It displays the steps, including identifying, formulating, solving, and answering. When entering the number of heads and feet, the number of feet must be even and between two and four times the number of heads. If the input is unreasonable, the program prompts for reentry.

```
How many heads? 36
How many legs? 126

If there are x rabbits, then there are 36 - x chickens.

     4x + 2·(36 - x) = 126
      4x + 2·36 - 2x = 126
             4x - 2x = 126 - 72
                  2x = 54
                   x = 54 / 2
                   x = 27

    Number of chickens: 36 - 27 = 9

So, there are 9 chickens and 27 rabbits.
```

Area of Polygons



Area of a parallelogram = base × height

$$S = ah$$

Area of a triangle = base × height ÷ 2

$$S = ah \div 2$$

Area of a trapezoid = (upper base + lower base) × height ÷ 2

$$S = (a + b)h \div 2$$

Area of composite figures

Estimating the area of irregular shapes

Program

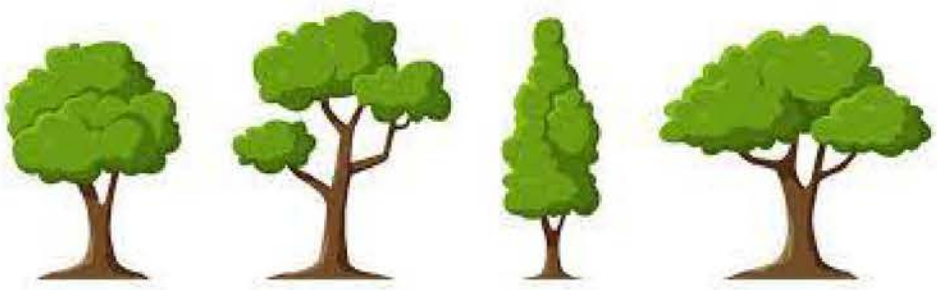
Polygon Classes with Area Properties

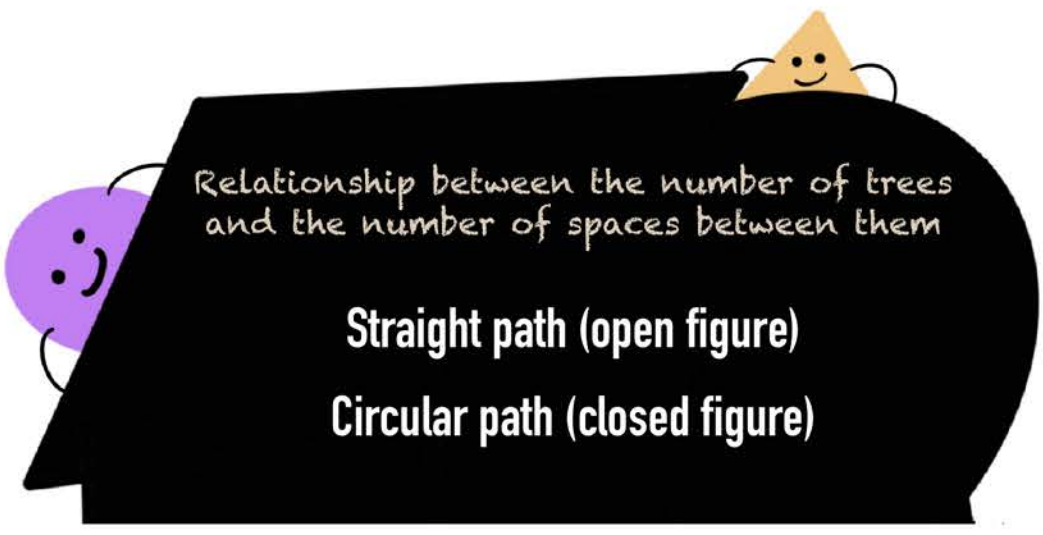
polygon_classes.py class

This program defines classes for parallelograms, triangles, and trapezoids. Each class has a property "area". The value of area is determined by other attributes such as base and height. If other attributes change, area will also change accordingly. The area attribute cannot be directly modified - such special attributes are called "properties" in Python. When a property is defined inside a class, the "@property" decorator is added before a function of the same name. Properties are accessed the same way as other attributes, but accessing a property is actually invoking a function internally. So, more can be done when accessing properties compared to attributes.

```
>>> parallelogram = Parallelogram(4, 3)
>>> print(parallelogram.area)
12
>>>
>>> parallelogram.area = 10
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
AttributeError: can't set attribute 'area'
>>>
>>> parallelogram.base = 5
>>> print(parallelogram)
Parallelogram
base: 5
height: 3
area: 15

>>>
```

Tree Planting Problem





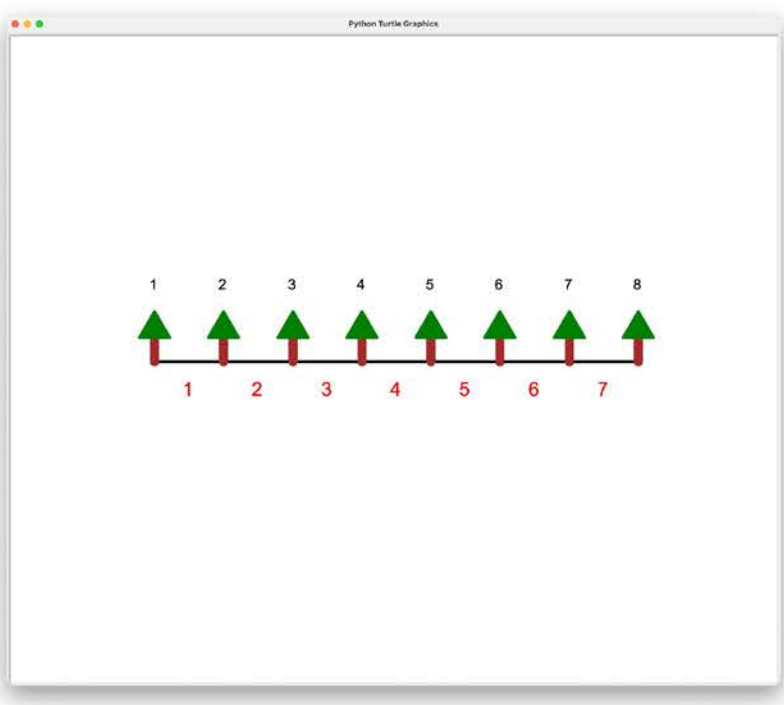
Program

Tree Planting Problem

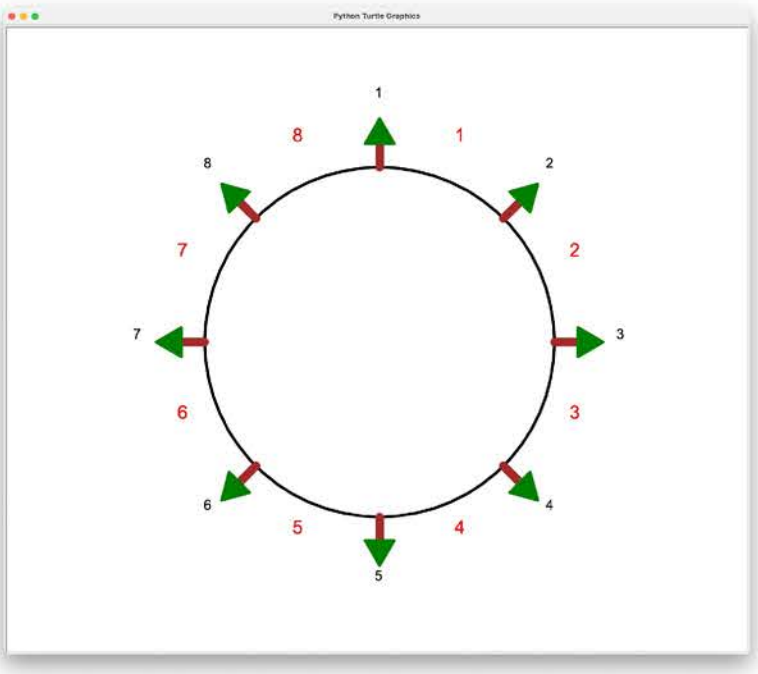
plant_trees.py turtle

In this program, the user enters the number of trees to plant and selects whether they want to plant the trees on a straight path (an open figure) or on a circular path (a closed figure). Based on the user's input, the program uses Turtle to draw the path and trees. It also displays the number of trees and the number of spaces between them.

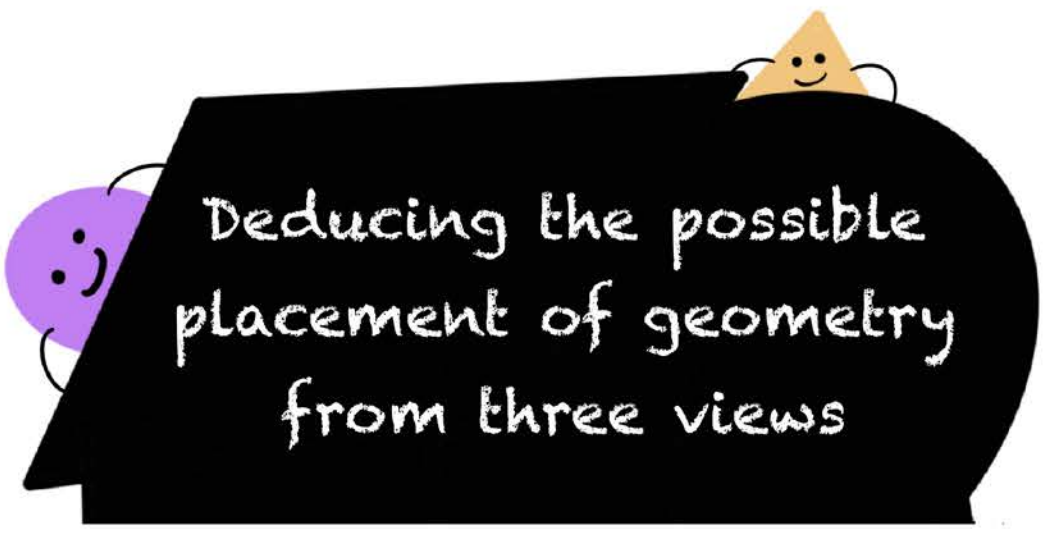
```
Number of trees (2-20): 8
Is the road a closed shape (y/n)? n
```




```
Number of trees (2-20): 8
Is the road a closed shape (y/n)? y
```



Observing Objects 2



Program

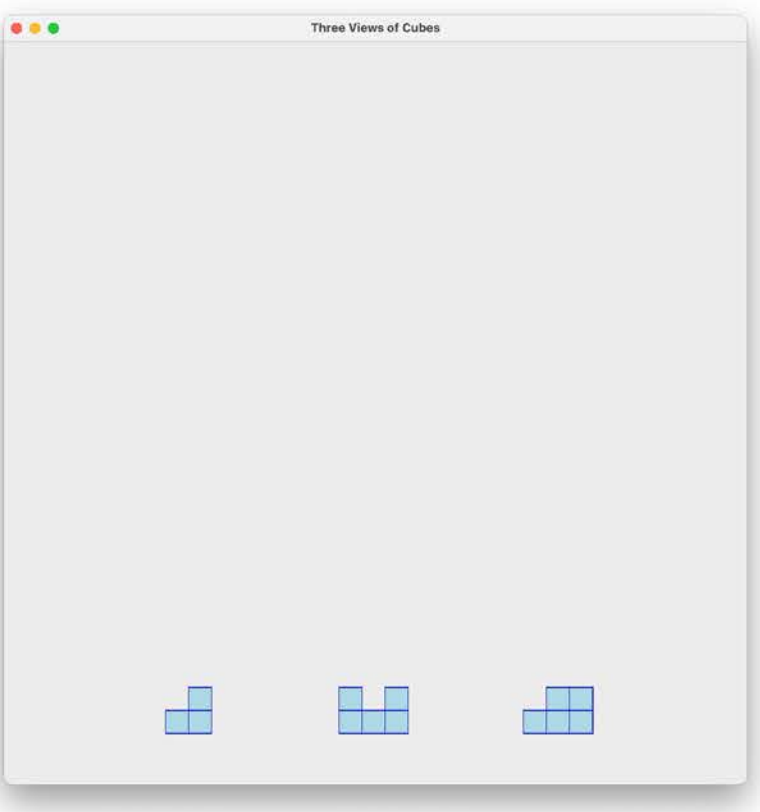
Three Views of Cubes v2

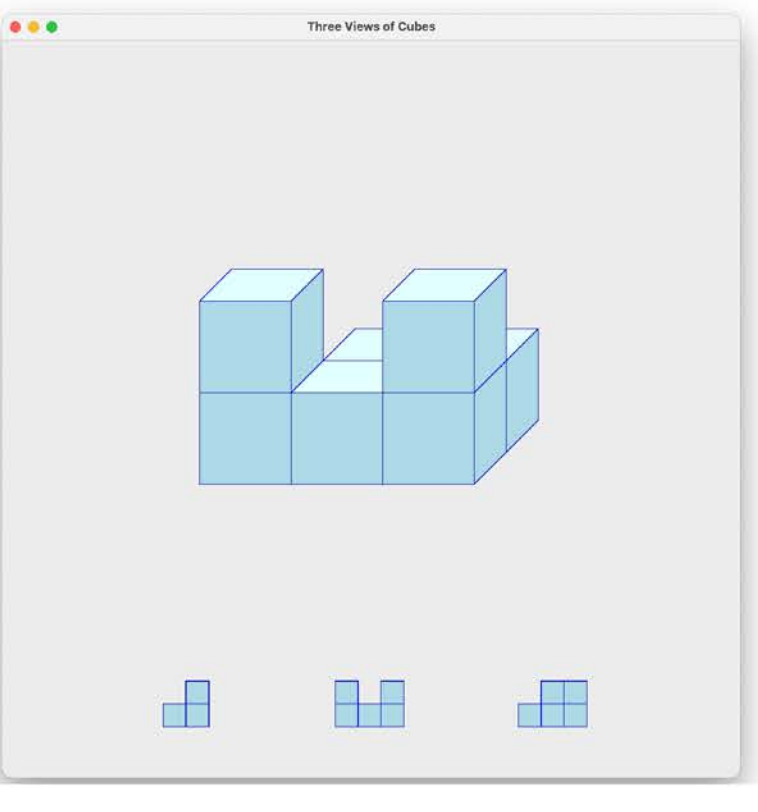
cubes_3view_v2.py

tkinter, coordinate, 2d list, random, class

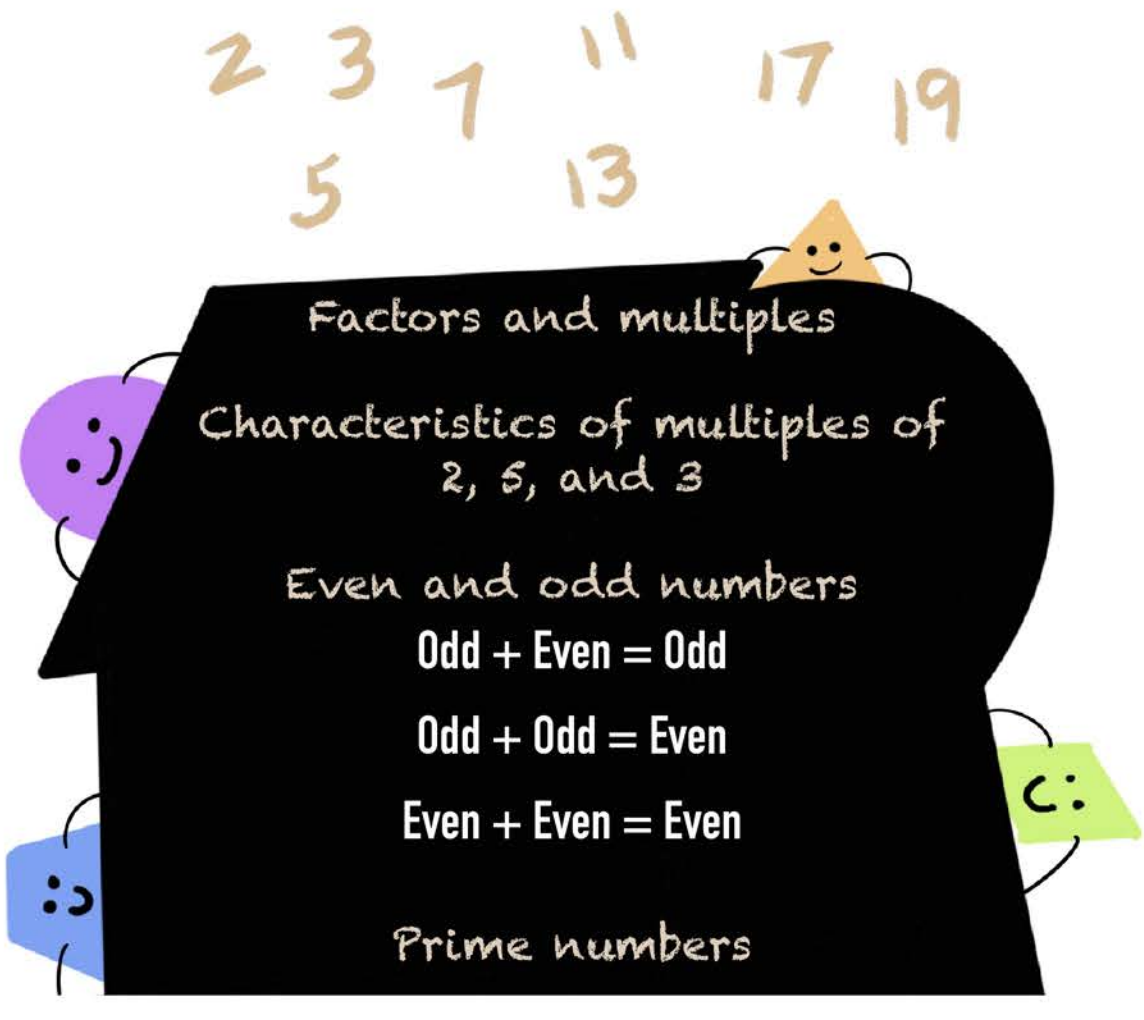
The program randomly generates an object made of cubes and then displays the three views of the object at the bottom of the window. From left to right, they are the left view, front view, and top view. Press the spacebar to display the object. Press the spacebar again to generate a new object.

The program is based on “Three Views of Cubes” (G422), which primarily focuses on the ability to draw three views of a given geometric figure. Therefore, it initially displayed the geometric figure, and pressing the spacebar would show the three views below the window. In this unit, the main learning objective is to derive the possible arrangements of geometric figures from three-view drawings. Therefore, the program first shows the three views below the window, allowing learners to think about the spatial structure of the geometric figure (with multiple possible answers). Pressing the spacebar would then display the geometric figure in the center of the window.





Factors and Multiples



Program 1

Get Prime Numbers

get_prime_numbers.py algorithm

There are 4 functions related to factors and prime numbers in the program.

1. get_factors(n): get all factors of n. If i is a factor of n, then n/i is also a factor of n. So, we don't need to find factors from 1 all the way to n or n/2. Instead, we just need to find factors from 1 to sqrt(n).

2. is_prime_number(n): check if n is a prime number. Besides 2, only odd numbers that are not larger than sqrt(n) need to be checked.

3. get_prime_numbers_slow(n): get all prime numbers within n by using is_prime_number() to check each number one by one. This method is slower.

4. get_prime_numbers(n): get all prime numbers within n by removing all multiples of numbers within the range. The numbers left are prime numbers. This method is faster. num_list is a list of all numbers up to n. The values represent which numbers are removed (False) and which numbers are left (True). The numbers left will be prime numbers. To find all prime numbers within n, we only need to remove numbers that are multiples of numbers from 2 to int(sqrt(n)). This is because numbers that are k times int(sqrt(n)) (k < sqrt(n)) have already been removed and numbers that are k times int(sqrt(n)) (k > sqrt(n)) exceed n.

```
Factors of 100 are: [1, 2, 4, 5, 10, 20, 25, 50, 100]

Prime numbers within 100 are: [2, 3, 5, 7, 11, 13, 17, 19, 23,
29, 31, 37, 41, 43, 47, 53, 59, 61, 67, 71, 73, 79, 83, 89, 97]

Number of prime numbers within 10000000: 664579
Running time for slow method: 28.06s

Number of prime numbers within 10000000: 664579
Running time for fast method: 1.11s
```

Program 2

Goldbach Conjecture

goldbach.py

The Goldbach Conjecture states that any even number greater than 2 can be expressed as the sum of two prime numbers. For a user-input even number greater than 2, the program expresses it as the sum of two prime numbers.

```
Enter an even number greater than 2: 98
98 = 19 + 79
```

Cuboids and Cubes

Cuboid: length, width, height

Cube: edge length

8 vertices, 12 edges, 6 faces

Surface area of cuboids and cubes

Surface area of a cuboid = (length × width + length × height + width × height) × 2

$$S = 2(ab + ah + bh)$$

Surface area of a cube = (edge length × edge length) × 6

$$S = 6a^2$$

Volume

Volume of cuboids and cubes

Volume of a cuboid = length × width × height

$$V = abh$$

Volume of cube = edge length × edge length × edge length

$$V = a^3$$

Volume of cuboid (or cube) = base area × height

$$V = Sh$$

Volume units (cubic centimeters, cubic decimeters, cubic meters)

1 Cubic Decimeter (dm^3) = 1000 Cubic Centimeters (cm^3)

1 Cubic Meter (m^3) = 1000 Cubic Decimeters (dm^3)

Volume units (liters, milliliters)

1 Liter (L) = 1000 Milliliters (mL)

1 Liter (L) = 1 Cubic Decimeter (dm^3)

1 Milliliter (mL) = 1 Cubic Centimeter (cm^3)

Measuring the volume of irregular objects by displacing water

program 1

★★★★☆

Cuboid Class with Unit Property

cuboid.py class, exception, decimal

Similar to the program "Polygon Classes with Area Attributes" (G516), this program features the Cuboid class with surface area and volume properties that are calculated based on length, width, height, or edge length. These properties are implemented with getters (with no setters) and cannot be directly modified.

The program also includes the unit property. When the unit is changed, the dimensions (length, width, height) adjust accordingly in the setter for the unit attribute. Both the unit's getter and setter use the same function name, "unit," with the "@property" decorator for the getter and the "@unit.setter" decorator for the setter.

When instantiating a Cuboid object, 0, 1, or 3 edge arguments can be given. If other numbers of edge arguments are given, or if any edge value is not valid, the program can still pass the syntax test, but errors will occur at runtime - such errors are called exceptions. Exceptions can be handled in programs. Exceptions that are not handled will result in error messages. In addition to built-in exceptions, there are also user-defined exceptions that can be raised manually with the "raise" keyword.

The program defines a custom exception for the wrong number of edge arguments. If there are any problems with instantiating a Cuboid object or setting a property, a corresponding built-in or user_defined exception will be raised, and the error message will be displayed.

```
>>> cuboid = Cuboid(3, 2, unit='dm')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/Users/felix/Desktop/Learn Math with Coding/g523_cuboid/cuboid.py",
line 71, in __init__
    raise NumberOfArgumentsError(len(edges))
cuboid.NumberOfArgumentsError: The number of edge arguments can only be 0,
1, or 3. You have 2 edge arguments.
>>>
>>> cuboid = Cuboid(3, unit='dm')
Successfully created a cube.
Edge: 3dm

>>> cuboid.height = 5
>>> print(cuboid)
Cuboid
Length: 3dm
Width: 3dm
Height: 5dm
Surface Area: 78dm²
Volume: 45dm³

>>> cuboid.unit = 'm'
>>> print(cuboid)
Cuboid
Length: 0.3m
Width: 0.3m
Height: 0.5m
Surface Area: 0.78m²
Volume: 0.045m³

>>>
```

Program 2

★★☆☆☆

Practice Volume Unit Conversion

volume_unit_conversion.py dict, exception, random

Randomly generate a specified number of volume unit conversion questions. Every time a question is answered, the program will display whether it is correct or incorrect. In the case of an incorrect answer, the correct one will be shown. The total score will be displayed after all questions are completed (the full score value can be set; the default is 100 points).

This program is based on "Practice Area Unit Conversion" (G412) and adds a new unit type volume to it with 4 volume units mm^3, cm^3, dm^3, and m^3. The program also uses a dictionary to store all unit types and provides a list for users to set which unit types they want to practice.

Users can input either decimals or fractions when converting from a smaller unit to a larger unit. When there are many digits to be entered, scientific notation can be used. E.g., 1e3 means 1 followed by three zeros, which is 1000; 1e-3 means 1/(1e3), which is 1/1000 or 0.001.

```
Question 1/4:  1mm³ = __m³
? 1e-9
Hooray!

Question 2/4:  1m³ = __dm³
? 1000
Super!

Question 3/4:  1cm³ = __dm³
? 0.001
Bravo!

Question 4/4:  1mm³ = __dm³
? 1/1000
Incorrect. The answer is 1e-06 or 1/1000000.

Your score is 75/100.
```

Meaning and Properties of Fractions

Meaning of fractions

The unit fraction

Fractions and division:

$$a \div b = \frac{a}{b} \ (b \neq 0)$$

Proper fractions and improper fractions: improper fractions can be converted into whole numbers or mixed numbers.

Prime factorization and short division

Basic properties of fractions: the value of a fraction remains the same when both the numerator and denominator are multiplied or divided by the same number (excluding 0).

Greatest common divisor

Least common multiple

Fraction simplification

Comparing fractions with different denominators

Fractions and decimals

Conversion between fractions and decimals

Program 1

Greatest Common Divisor and Least Common Multiple

gcd_lcm.py algorithm

This program uses two methods to find the greatest common divisor (GCD) of two numbers, with the larger number denoted as "a" and the smaller number as "b."

The first method employs a loop to sequentially check if numbers smaller than "b" are common factors of both numbers, eventually identifying the greatest common factor among all the common factors. The optimized number of iterations is from 1 to sqrt(b), similar to the get_factors() function in “Get Prime Numbers” (G522). However, even with optimized iterations, the program may still take a long time to execute when dealing with large numbers.

The second method employs Euclid's division algorithm. The algorithm is based on the principle that the greatest common factor of two integers is the same as the greatest common factor of the smaller number and the remainder of their division. The algorithm works as follows:

1. Divide the larger number by the smaller number, with the larger number as the dividend and the smaller number as the divisor. To find the greatest common factor of the dividend and divisor, only the divisor and the remainder of their division need to be considered.
2. Repeat the first step using the divisor as the new dividend and the remainder as the new divisor.
3. Continue this process until the remainder becomes 0. At this point, the divisor is the greatest common factor of the original two numbers.

The Euclidean algorithm is highly efficient at reducing two large numbers quickly, making it ideal for finding the greatest common factor. The program also utilizes the GCD to determine the least common multiple (LCM), as the product of two numbers is equal to the product of their GCD and LCM.

```
The GCD of 121932630989178480 and 121932631112635269 is 123456789.
Running time for slow method: 13.52s

The GCD of 121932630989178480 and 121932631112635269 is 123456789.
Running time for fast method: 0.0s

The LCM of 36 and 48 is 144.
```

Program 2

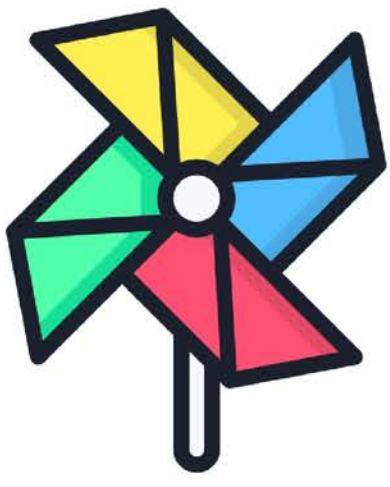
Convert Decimal to Simplest Fraction

decimal_to_fraction.py string

In this program, the user inputs a decimal number, and the program converts it into the simplest fraction, displaying the result on the screen. To simplify the fraction, the program uses the greatest common divisor (GCD) calculation function from Program 1 to obtain the GCD of the numerator and denominator.

```
Enter a decimal: 0.48
0.48 = 12/25
```

Rotation



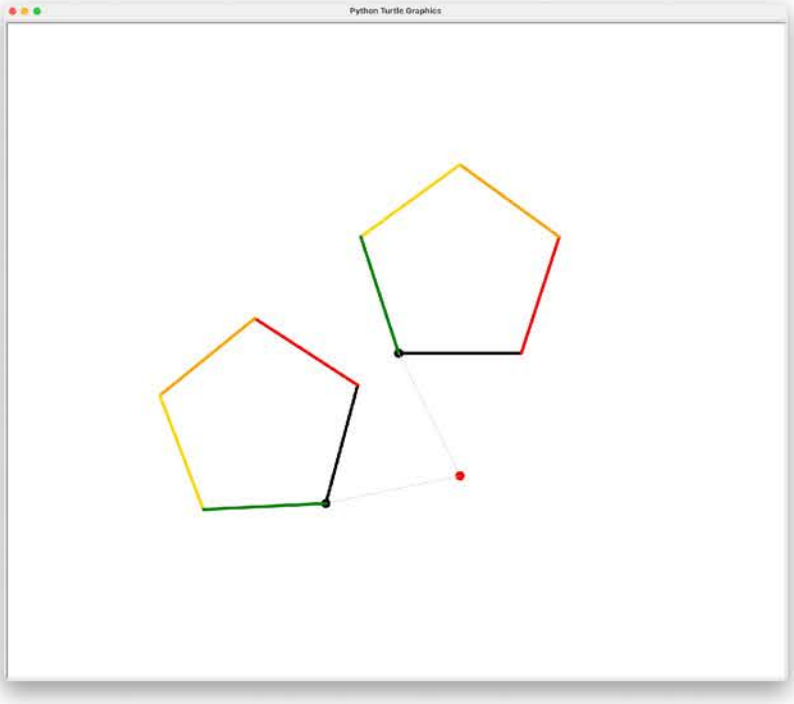
Clockwise and counterclockwise rotation of shapes around a point

Program

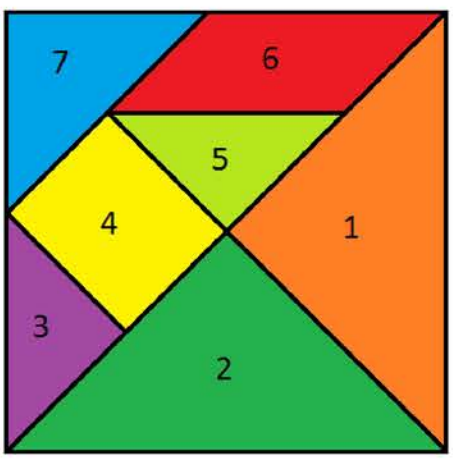
Rotation

rotation.py turtle, coordinate, random

This program randomly generates a regular polygon and rotates it around a specified point and by a given rotation angle. To distinguish between the edges of the shape before and after rotation, each edge of the shape is assigned a different color.



Addition and Subtraction of Fractions



$$\frac{1}{4}+\frac{1}{4}+\frac{1}{16}+\frac{1}{8}+\frac{1}{16}+\frac{1}{8}+\frac{1}{8}=1$$

Program

Addition and Subtraction of Fractions

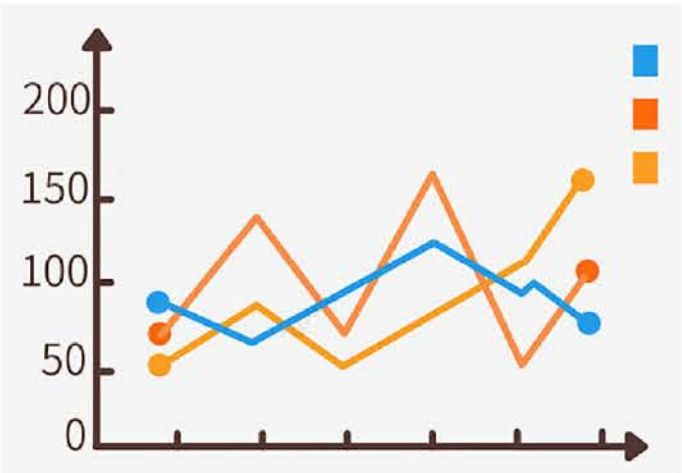
add_sub_fractions.py string

At the beginning of the program, the user is prompted to input two fractions or integers and choose between addition or subtraction. The program then calculates and displays the result in its simplest form.

When performing fraction addition and subtraction, the program calls the function for finding the least common multiple from the program "Greatest Common Divisor and Least Common Multiple" (G524). It first converts fractions with different denominators to fractions with the same denominator. After performing addition or subtraction with common denominators, it uses the fraction simplification function from the program "Convert Decimal to Simplest Fraction" (G524) to simplify the result.

```
Enter the first fraction: 1/3
Enter the second fraction: 1/6
Choose between addition and subtraction (+ or -)? +
1/3 + 1/6 = 1/2

Enter the first fraction: 3/5
Enter the second fraction: 1
Choose between addition and subtraction (+ or -)? -
1 - 3/5 = 2/5
```

Line Charts



Program

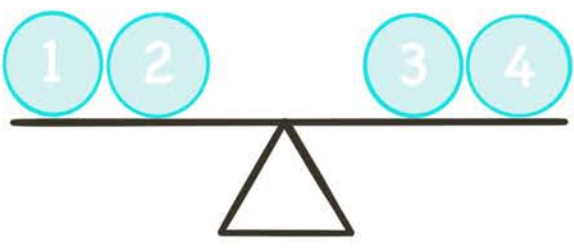
Improve Data Class to Draw Multi-Line Charts

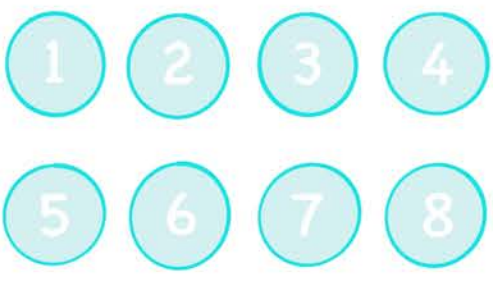
data3.py matplotlib, class, 2d list

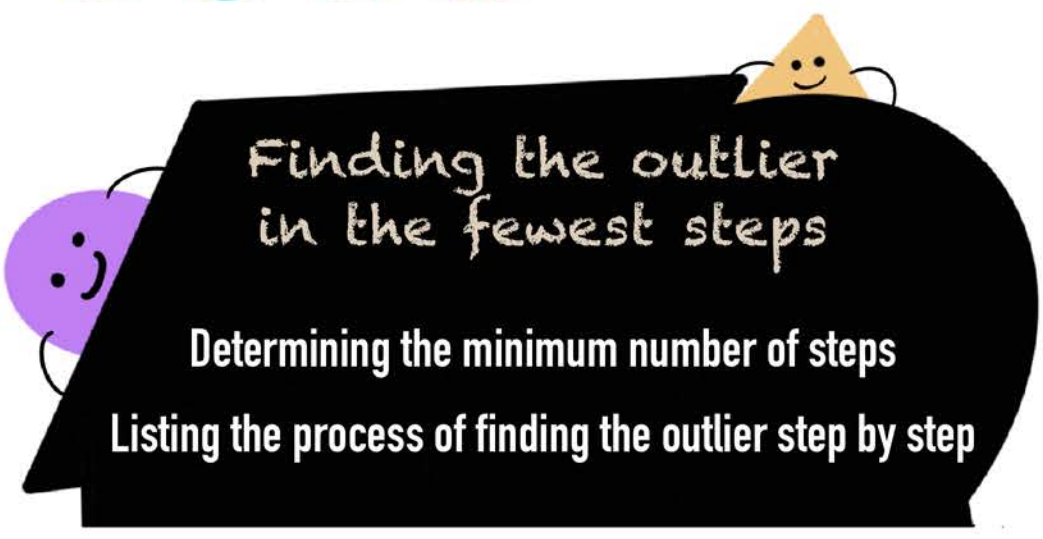
The program adds the line() method for drawing multi-line charts to the Data class in “Mean Value and Grouped Bar Charts” (G428). Except for the newly added method, the rest of the code in the Data class remains unchanged.

	Sun	Mon	Tue	Wed	Thu	Fri	Sat
Math	3	1	0	2	1	3	2
Music	3	2	2	2	1	3	1
P.E.	0	1	2	3	0	0	0

Identify the Outlier







Program

Identify the Outlier

identify_outlier.py recursion, algorithm, string

In this program, the user first enters the number of items that need to be checked, and the program displays the entire process of finding the outlier (whose weight is different) from these items. The challenge of the program lies in the fact that each weighing has two scenarios: balanced and unbalanced. In each scenario, there are again balanced and unbalanced weighings, and so on, creating a nested, branching structure for the entire process. This is different from the common program flow with sequential, conditional, loop, or function call statements.

The program uses recursion - an algorithm that solves a problem by solving a smaller instance of the same problem until the problem is so small that it can be solved directly (this smallest problem is known as the base case). Specifically, recursion uses functions that call themselves from within their own code. When using recursion, it is essential to provide termination conditions (base cases). Otherwise, the program will result in infinite recursion and end up exceeding the maximum recursion depth (default is 1000 in Python).

```
How many items need to be checked? 20
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20]

Round 1: [1, 2, 3, 4, 5, 6, 7] vs [8, 9, 10, 11, 12, 13, 14]
If balanced:
....Round 2: [15, 16] vs [17, 18]
....If balanced:
........Round 3: [19] vs [20]
........The defective item is identified.
....If unbalanced, assuming [15, 16] contains the outlier:
........Round 3: [15] vs [16]
........The defective item is identified.
If unbalanced, assuming [1, 2, 3, 4, 5, 6, 7] contains the outlier:
....Round 2: [1, 2] vs [3, 4]
....If balanced:
........Round 3: [5] vs [6]
........If balanced, 7 contains the outlier.
........If unbalanced, the defective item is identified.
....If unbalanced, assuming [1, 2] contains the outlier:
........Round 3: [1] vs [2]
........The defective item is identified.

We can ensure that we will find the defective item in at least 3 rounds.
```

Acknowledgments

HAPPY LEARNING!

I would like to express my gratitude to my brother Henry for his active collaboration throughout the entire project.

I extend my thanks to the mentors and friends who have provided me with guidance and assistance in various academic fields, especially my dad, who first introduced me to mathematics and programming.

Special appreciation goes to my mom for providing illustrations for this handbook.

Email: math-coding@hotmail.com

Project code is on GitHub