# Programming Challenges for IB – (Java)

## Introduction to Programming Exercises.

In order to complete these exercises you must be able to…

* Create a new Java project with *public static void main*
* Print strings and variables
* Declare and initialize *static global* variables for all the fundamental data types
* Manipulate variables using basic mathematic operators

**Exercise 1: Mailing Address**

Create a program that displays your name and complete mailing address formatted in the manner that you would usually see it on the outside of an envelope. Your program does not need to read any input from the user.

**Exercise 2: Hello**

Write a program that stores the users first and last name into two separate variables. The program should respond with a message that says hello to the user, using his or her name.

**Exercise 3: Area of a Room**

Write a program stores the height, length and width as a room as variables. Your program should compute and display the area of the room. The length and the width will be entered as floating point numbers. Include units in your prompt and output message in meters

**Exercise 4: Area of a Field**

Create a program that stores the length and width of a farmer’s field from the user in feet into two separate variables. Display the area of the field in acres.

**Exercise 5: Bottle Deposits**

In many jurisdictions a small deposit is added to drink containers to encourage people to recycle them. In one particular jurisdiction, drink containers holding one liter or less have a $0.10 deposit, and drink containers holding more than one liter have a $0.25 deposit.

Write a program that stores a number from the user into a variable. Your program should continue by computing and displaying the refund that will be received for returning those containers. You do not have to round the output to any level of significance.

In order to complete these tasks you need to be able to…

* Use the scanner to read a variable from the user
* Typecast datatypes

**Exercise 6:Tax and Tip**

The program that you create for this exercise will begin by reading the cost of a meal ordered at a restaurant from the user. Then your program will compute the tax and tip for the meal. Use your local tax rate when computing the amount of tax owing. Compute the tip as 18 percent of the meal amount (without the tax). The output from your program should include the tax amount, the tip amount, and the grand total for the meal including both the tax and the tip. Format the output so that all of the values are displayed using two decimal places.

**Exercise 7:Sum of the First *n* Positive Integers**

Write a program that reads a positive integer, *n*, from the user and then displays the sum of all of the integers from 1 to *n*. The sum of the first *n* positive integers can be computed using the formula:

A picture containing object

Description automatically generated

**Exercise 8:Widgets and Gizmos**

An online retailer sells two products: widgets and gizmos. Each widget weighs 75 grams. Each gizmo weighs 112 grams. Write a program that reads the number of widgets and the number of gizmos in an order from the user. Then your program should compute and display the total weight of the order.

**Exercise 9: Compound Interest**

Pretend that you have just opened a new savings account that earns 4 percent interest per year. The interest that you earn is paid at the end of the year, and is added to the balance of the savings account. Write a program that begins by reading the amount of money deposited into the account from the user. Then your program should compute and display the amount in the savings account after 1, 2, and 3 years. Display

each amount so that it is rounded to 2 decimal places.

**Exercise 10:Arithmetic**

Create a program that reads two integers, *a* and *b*, from the user. Your program should compute and display:

• The sum of *a* and *b*

• The difference when *b* is subtracted from *a*

• The product of *a* and *b*

• The quotient when *a* is divided by *b*

• The remainder when *a* is divided by *b*

• The result of log10 *a*

• The result of *ab*

You are expected to research online to find out how to complete some of these calculations within Java.

**Exercise 11: Fuel Efficiency**

In the United States, fuel efficiency for vehicles is normally expressed in miles-per gallon(MPG). In Canada, fuel efficiency is normally expressed in liters-per-hundred kilometers (L/100 km). Use your research skills to determine how to convert from MPGto L/100 km. Then create a program that reads a value from the user in American units and displays the equivalent fuel efficiency in Canadian units.

**Exercise 12: Distance Between Two Points on Earth**

The surface of the Earth is curved, and the distance between degrees of longitude varies with latitude. As a result, finding the distance between two points on the surface of the Earth is more complicated than simply using the Pythagorean theorem.

Let (t1, g1) and (t2, g2) be the latitude and longitude of two points on the Earth’s surface. The distance between these points, following the surface of the Earth, in kilometers is:

A close up of a logo

Description automatically generated

The value 6371.01 in the previous equation wasn’t selected at random. It is the average radius of the Earth in kilometers.

Create a program that allows the user to enter the latitude and longitude of two points on the Earth in degrees. Your program should display the distance between the points, following the surface of the earth, in kilometers.

**Exercise 13:Making Change**

Consider the software that runs on a self-checkout machine. One task that it must be able to perform is to determine how much change to provide when the shopper pays for a purchase with cash. Write a program that begins by reading a number of cents from the user as an integer. Then your program should compute and display the denominations of the coins that should be used to give that amount of change to the shopper. The change should be given using as few coins as possible. Assume that the machine is loaded with pennies, nickels, dimes, quarters, loonies and toonies.

A crowd of people

Description automatically generated

**Exercise 14: Height Units**

Many people think about their height in feet and inches, even in some countries that primarily use the metric system. Write a program that reads a number of feet from the user, followed by a number of inches. Once these values are read, your program should compute and display the equivalent number of centimeters.

A close up of a mans face

Description automatically generated

**Exercise 15: Distance Units**

In this exercise, you will create a program that begins by reading a measurementin feet from the user. Then your program should display the equivalent distance in inches, yards and miles. Use the Internet to look up the necessary conversion factors if you don’t have them memorized.

## If statements

In order to complete these exercises you must be able to…

• Make a decision with an if statement

• Select one of two alternatives with an if-else statement

• Select from one of several alternatives by using an if-elseif or if-elseif-else statement

• Construct a complex condition for an if statement that includes the Boolean operators

and, or and not

• Nest an if statement within the body of another if statement

**Exercise 34: Even or Odd?**

Write a program that reads an integer from the user. Then your program should display a message indicating whether the integer is even or odd.

**Exercise 35: Dog Years**

It is commonly said that one human year is equivalent to 7 dog years. However this simple conversion fails to recognize that dogs reach adulthood in approximately two years. As a result, some people believe that it is better to count each of the first two human years as 10.5 dog years, and then count each additional human year as 4 dog years.

Write a program that implements the conversion from human years to dog years described in the previous paragraph. Ensure that your program works correctly for conversions of less than two human years and for conversions of two or more human years. Your program should display an appropriate error message if the user enters a negative number.

**Exercise 36:Vowel or Consonant**

In this exercise you will create a program that reads a letter of the alphabet from the user. If the user enters a, e, i, o or u then your program should display a message indicating that the entered letter is a vowel. If the user enters y then your program should display a message indicating that sometimes y is a vowel, and sometimes y is a consonant. Otherwise your program should display a message indicating that the letter is a consonant.

**Exercise 37:Name that Shape**

Write a program that determines the name of a shape from its number of sides. Read the number of sides from the user and then report the appropriate name as part of a meaningful message. Your program should support shapes with anywhere from 3 up to (and including) 10 sides. If a number of sides outside of this range is entered then your program should display an appropriate error message.

**Exercise 38:Month Name to Number of Days**

The length of a month varies from 28 to 31 days. In this exercise you will create a program that reads the name of a month from the user as a string. Then your program should display the number of days in that month. Display “28 or 29 days” for February so that leap years are addressed.

**Exercise 39: Sound Levels**

The following table lists the sound level in decibels for several common noises. A screenshot of a cell phone

Description automatically generated

Write a program that reads a sound level in decibels from the user. If the user enters a decibel level that matches one of the noises in the table then your program should display a message containing only that noise. If the user enters a number of decibels between the noises listed then your program should display a message indicating which noises the level is between. Ensure that your program also generates reasonable output for a value smaller than the quietest noise in the table, and for a value larger than the loudest noise in the table.

**Exercise 40:Name that Triangle**

A triangle can be classified based on the lengths of its sides as equilateral, isosceles or scalene. All 3 sides of an equilateral triangle have the same length. An isosceles triangle has two sides that are the same length, and a third side that is a different length. If all of the sides have different lengths then the triangle is scalene.

Write a program that reads the lengths of 3 sides of a triangle from the user. Display a message indicating the type of the triangle.

**Exercise 41: Note To Frequency**

The following table lists an octave of music notes, beginning with middle C, along with their frequencies.

A screenshot of a cell phone

Description automatically generated

Begin by writing a program that reads the name of a note from the user and displays the note’s frequency. Your program should support all of the notes listed previously.

Once you have your program working correctly for the notes listed previously you should add support for all of the notes from C0 to C8. While this could be done by adding many additional cases to your if statement, such a solution is cumbersome, inelegant and unacceptable for the purposes of this exercise. Instead, you should exploit the relationship between notes in adjacent octaves. In particular, the frequency of any note in octave *n* is half the frequency of the corresponding note in octave *n*+1.

By using this relationship, you should be able to add support for the additional notes without adding additional cases to your if statement.

A screenshot of a cell phone

Description automatically generated

**Exercise 42: Frequency To Note**

In the previous question you converted from note name to frequency. In this question you will write a program that reverses that process. Begin by reading a frequency from the user. If the frequency is within one Hertz of a value listed in the table in the previous question then report the name of the note. Otherwise report that the frequency does not correspond to a known note. In this exercise you only need to consider the notes listed in the table. There is no need to consider notes from other octaves.

**Exercise 43: Faces onMoney**

It is common for images of a country’s previous leaders, or other individuals of historical significance, to appear on its money. The individuals that appear on banknotes

in the United States are listed in table below.

Write a program that begins by reading the denomination of a banknote from the user. Then your program should display the name of the individual that appears on the bank note of the entered amount. An appropriate error message should be displayed if no such note exists.

A screenshot of a cell phone

Description automatically generated

A screenshot of a cell phone

Description automatically generated

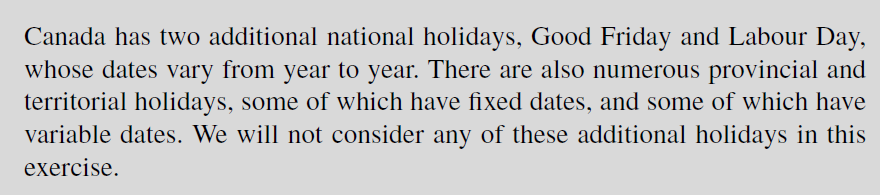
**Exercise 44: Date to Holiday Name**

(*18 Lines*)

Canada has three national holidays which fall on the same dates each year.

A screenshot of a cell phone

Description automatically generated

Write a program that reads a month and day from the user. If the month and day match one of the holidays listed previously then your program should display the holiday’s name. Otherwise your program should indicate that the entered month an day do not correspond to a fixed-date holiday.

**Exercise 45:What Color is that Square?**

Positions on a chess board are identified by a letter and a number. The letter identifies the column, while the number identifies the row, as shown below:A picture containing crossword puzzle, text

Description automatically generated

Write a program that reads a position from the user. Use an if statement to determine if the column begins with a black square or a white square. Then use modular arithmetic to report the color of the square in that row. For example, if the user enters a1 then your program should report that the square is black. If the user enters d5 then your program should report that the square is white. Your program may assume that a valid position will always be entered. It does not need to perform any error checking.

**Exercise 46: Season from Month and Day**

The year is divided into four seasons: spring, summer, fall and winter. While the exact dates that the seasons change vary a little bit from year to year because of the way that the calendar is constructed, we will use the following dates for this exercise:A screenshot of a cell phone

Description automatically generated

Create a program that reads a month and day from the user. The user will enter the name of the month as a string, followed by the day within the month as an integer. Then your program should display the season associated with the date that was entered.

**Exercise 47: Birth Date to Astrological Sign**

The horoscopes commonly reported in newspapers use the position of the sun at the time of one’s birth to try and predict the future. This system of astrology divides the year into twelve zodiac signs, as outline in the table below:A screenshot of a cell phone

Description automatically generated

Write a program that asks the user to enter his or her month and day of birth. Then your program should report the user’s zodiac sign as part of an appropriate output message.

**Exercise 48: Chinese Zodiac**

The Chinese zodiac assigns animals to years in a 12 year cycle. One 12 year cycle is shown in the table below. The pattern repeats from there, with 2012 being another year of the dragon, and 1999 being another year of the hare.A screenshot of a cell phone

Description automatically generated

Write a program that reads a year from the user and displays the animal associated with that year. Your program should work correctly for any year greater than or equal to zero, not just the ones listed in the table.

**Exercise 49: Richter Scale**

The following table contains earthquake magnitude ranges on the Richter scale and their descriptors:

A screenshot of a cell phone

Description automatically generated

Write a program that reads a magnitude from the user and displays the appropriate descriptor as part of a meaningful message. For example, if the user enters 5.5 then your program should indicate that a magnitude 5.5 earthquake is considered to be a moderate earthquake.

**Exercise 61:Average**

In this exercise you will create a program that computes the average of a collection of values entered by the user. The user will enter 0 as a sentinel value to indicate that no further values will be provided. Your program should display an appropriate error message if the first value entered by the user is 0.

A close up of a logo

Description automatically generated

**Exercise 62: Discount Table**

A particular retailer is having a 60 percent off sale on a variety of discontinued products. The retailer would like to help its customers determine the reduced price of the merchandise by having a printed discount table on the shelf that shows the original prices and the prices after the discount has been applied. Write a program that uses a loop to generate this table, showing the original price, the discount amount, and the new price for purchases of $4.95, $9.95, $14.95, $19.95 and $24.95.

**Exercise 63: Temperature Conversion Table**

Write a program that displays a temperature conversion table for degrees Celsius and degrees Fahrenheit. The table should include rows for all temperatures between 0 and 100 degrees Celsius that are multiples of 10 degrees Celsius. Include appropriate headings on your columns. The formula for converting between degrees Celsius and degrees Fahrenheit can be found on the internet.

**Exercise 64: No More Pennies**

February 4, 2013 was the last day that pennies were distributed by the Royal Canadian Mint. Now that pennies have been phased out retailers must adjust totals so that they are multiples of 5 cents when they are paid for with cash (credit card and debit card transactions continue to be charged to the penny). While retailers have some freedom in how they do this, most choose to round to the closest nickel.

Write a program that reads prices from the user until a blank line is entered. Display the total cost of all the entered items on one line, followed by the amount due if the customer pays with cash on a second line. The amount due for a cash payment should be rounded to the nearest nickel. One way to compute the cash payment amount is to begin by determining how many pennies would be needed to pay the total. Then compute the remainder when this number of pennies is divided by 5. Finally, adjust the total down if the remainder is less than 2.5. Otherwise adjust the total up.

**Exercise 65: Compute the Perimeter of a Polygon**

Write a program that computes the perimeter of a polygon. Begin by reading the x and y values for the ﬁrst point on the perimeter of the polygon from the user. Then continue reading pairs of x and y values until the user enters a blank line for the x-coordinate. Each time you read an additional coordinate you should compute the distance to the previous point and add it to the perimeter. When a blank line is entered for the x-coordinate your program should add the distance from the last point back to the ﬁrst point to the perimeter. Then it should display the total perimeter. Sample input and output is shown below, with user input shown in bold:

Enter the x part of the coordinate: 0

Enter the y part of the coordinate: 0

Enter the x part of the coordinate: (blank to quit): 1

Enter the y part of the coordinate: 0

Enter the x part of the coordinate: (blank to quit): 0

Enter the y part of the coordinate: 1

Enter the x part of the coordinate: (blank to quit):

The perimeter of that polygon is 3.414213562373095

**Exercise 66: Compute a Grade Point Average**

Exercise 51 included a table that shows the conversion from letter grades to grade points at an academic institution. In this exercise you will compute the grade point average of an arbitrary number of letter grades entered by the user. The user will enter a blank line to indicate that all the grades have been provided. For example, if the user enters A, followed by C+, followed by B, followed by a blank line then your program should report a grade point average of 3.1.

You may ﬁnd your solution to Exercise 51 helpful when completing this exercise. Your program does not need to do any error checking. It can assume that each value entered by the user will always be a valid letter grade or a blank line.

**Exercise 67: Admission Price**

A zoo determines the price of admission based on the age of the guest. Guests 2 years of age and less are admitted without charge. Children between 3 and 12 years of age cost $14.00. Seniors aged 65 years and over cost $18.00. Admission for all other guests is $23.00.

Create a program that begins by reading the ages of all the guests in a group from the user, with one age entered on each line. The user will enter a blank line to indicate that there are no more guests in the group. Then your program should display the admission cost for the group with an appropriate message. The cost should be displayed using two decimal places.

**Exercise 68: Parity Bits**

A parity bit is a simple mechanism for detecting errors in data transmitted over an unreliable connection such as a telephone line. The basic idea is that an additional bit is transmitted after each group of 8 bits so that a single bit error in the transmission can be detected.

Parity bits can be computed for either even parity or odd parity. If even parity is selected then the parity bit that is transmitted is chosen so that the total number of one bits transmitted (8 bits of data plus the parity bit) is even. When odd parity is selected the parity bit is chosen so that the total number of one bits transmitted is odd.

Write a program that computes the parity bit for groups of 8 bits entered by the user using even parity. Your program should read strings containing 8 bits until the user enters a blank line. After each string is entered by the user your program should display a clear message indicating whether the parity bit should be 0 or 1. Display an appropriate error message if the user enters something other than 8 bits.

A screenshot of a cell phone

Description automatically generated

**Exercise 69: Approximate *π***

The value of *π* can be approximated by the following infinite series:

A screenshot of a video game

Description automatically generated

Write a program that displays 15 approximations of *π*. The first approximation

Should make use of only the first term from the infinite series. Each additional approximation

displayed by your program should include one more term in the series, making

it a better approximation of *π* than any of the approximations displayed previously.

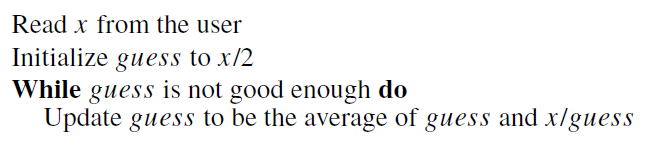
**Exercise 70: Caesar Cipher**

One of the first known examples of encryption was used by Julius Caesar. Caesar needed to provide written instructions to his generals, but he didn’t want his enemies to learn his plans if the message slipped into their hands. As result, he developed what later became known as the Caesar Cipher. the idea behind this cipher is simple (and as a result, it provides no protection against modern code breaking techniques). Each letter in the original message is shifted by 3 places. As a result, A becomes D, B becomes E, C becomes F, D becomes G, etc. The last three letters in the alphabet are wrapped around to the beginning: X becomes A, Y becomes B and Z becomes C. Non-letter characters are not modified by the cipher.

Write a program that implements a Caesar cipher. Allow the user to supply the message and the shift amount, and then display the shifted message. Ensure that your program encodes both uppercase and lowercase letters. Your program should also support negative shift values so that it can be used both to encode messages and decode messages.

**Exercise 71: Square Root**

Write a program that implements Newton’s method to compute and display the square root of a number entered by the user. The algorithm for Newton’s method follows:



When this algorithm completes, *guess* contains an approximation of the square root. The quality of the approximation depends on how you define “good enough”. In the author’s solution, *guess* was considered good enough when the absolute value of the difference between *guess* ∗ *guess* and *x* was less than or equal to 10−12.

**Exercise 72: Is a String a Palindrome?**

A string is a palindrome if it is identical forward and backward. For example “anna”, “civic”, “level” and “hannah” are all examples of palindromic words. Write a program that reads a string from the user and uses a loop to determines whether or not it is a palindrome. Display the result, including a meaningful output message.

**Exercise 73: Multiple Word Palindromes**

There are numerous phrases that are palindromes when spacing is ignored. Examples include “go dog”, “flee to me remote elf” and “some men interpret nine memos”, among many others. Extend your solution to exercise 72 so that it ignores spacing while determining whether or not a string is a palindrome. For an additional challenge, extend your solution so that is also ignores punctuation marks and treats uppercase

and lowercase letters as equivalent.

**Exercise 74: Multiplication Table**

In this exercise you will create a program that displays a multiplication table that shows the products of all combinations of integers from 1 times 1 up to and including 10 times 10. Your multiplication table should include a row of labels across the top of it containing the numbers 1 through 10. It should also include labels down the left side consisting of the numbers 1 through 10. The expected output from the program

is shown below:

A screenshot of a cell phone

Description automatically generated

**Exercise 75: Greatest Common Divisor**

The greatest common divisor of two positive integers, *n* and *m*, is the largest number, *d*, which divides evenly into both *n* and *m*. There are several algorithms that can be used to solve this problem, including:

Initialize *d* to the smaller of *m* and *n*. **While** *d* does not evenly divide *m* or *d* does not evenly divide *n* **do**

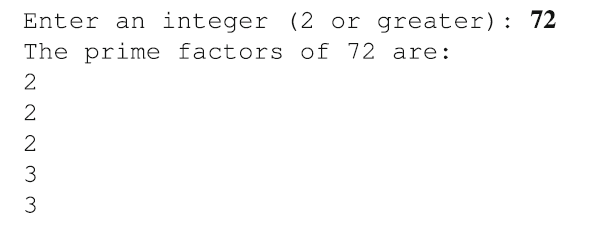
Decrease the value of *d* by 1 Report *d* as the greatest common divisor of *n* and *m* Write a program that reads two positive integers from the user and uses this algorithm to determine and report their greatest common divisor.

**Exercise 76: Prime Factors**

The prime factorization of an integer, *n*, can be determined using the following steps:A close up of a sign

Description automatically generated

Write a program that reads an integer from the user. If the value entered by the user is less than 2 then your program should display an appropriate error message. Otherwise your program should display the prime numbers that can be multiplied together to compute *n*, with one factor appearing on each line. For example:



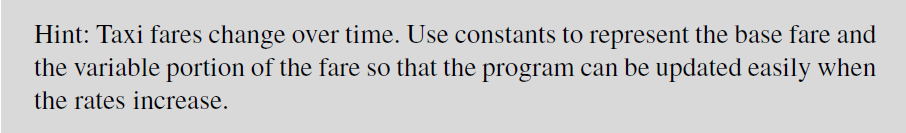
## Methods

**Exercise 81: Compute the Hypotenuse**

Write a function that takes the lengths of the two shorter sides of a right triangle as its parameters. Return the hypotenuse of the triangle, computed using Pythagorean theorem, as the function’s result. Include a main program that reads the lengths of the shorter sides of a right triangle from the user, uses your function to compute the length of the hypotenuse, and displays the result.

**Exercise 82:Taxi Fare**

In a particular jurisdiction, taxi fares consist of a base fare of $4.00, plus $0.25 for every 140 meters travelled. Write a function that takes the distance travelled (in kilometres) as its only parameter and returns the total fare as its only result. Write a main program that demonstrates the function.

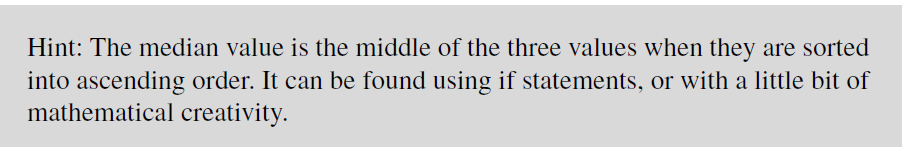


**Exercise 83: Shipping Calculator**

An online retailer provides express shipping for many of its items at a rate of $10.95 for the first item, and $2.95 for each subsequent item. Write a function that takes the number of items in the order as its only parameter. Return the shipping charge for the order as the function’s result. Include a main program that reads the number of items purchased from the user and displays the shipping charge.

**Exercise 84: Median of Three Values**

Write a function that takes three numbers as parameters and returns the median value of those parameters as its result. Include a main program that reads three values from the user and displays their median.

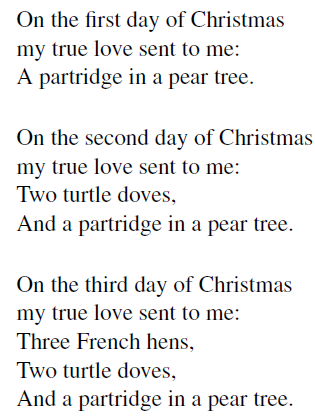


**Exercise 85: Convert an Integer to its Ordinal Number**

Words like first, second and third are referred to as ordinal numbers. In this exercise, you will write a function that takes an integer as its only parameter and returns a string containing the appropriate English ordinal number as its only result. Your function must handle the integers between 1 and 12 (inclusive). It should return an empty string if a value outside of this range is provided as a parameter. Include a main program that demonstrates your function by displaying each integer from 1 to 12 and its ordinal number. Your main program should only run when your file has not been imported into another program.

**Exercise 86:The Twelve Days of Christmas**

The Twelve Days of Christmas is a repetitive song that describes an increasingly long list of gifts sent to one’s true love on each of 12 days. A single gift is sent on the first day. A new gift is added to the collection on each additional day, and then the complete collection is sent. The first three verses of the song are shown below. The complete lyrics are available on the internet.



Your task is to write a program that displays the complete lyrics for The Twelve Days of Christmas. Write a function that takes the verse number as its only parameter and displays the specified verse of the song. Then call that function 12 times with integers that increase from 1 to 12.

Each item that is sent to the recipient in the song should only appear once in your program, with the possible exception of the partridge. It may appear twice if that helps you handle the difference between “A partridge in a pear tree” in the first verse and “And a partridge in a pear tree” in the subsequent verses. Import your solution to Exercise 85 to help you complete this exercise.

**Exercise 87: Centre a String in the Terminal**

Write a function that takes a string of characters as its first parameter, and the width of the terminal in characters as its second parameter. Your function should return a new string that consists of the original string and the correct number of leading spaces so that the original string will appear cantered within the provided width when it is printed. Do not add any characters to the end of the string. Include a main program that demonstrates your function.

**Exercise 88: Is it a Valid Triangle?**

If you have 3 straws, possibly of differing lengths, it may or may not be possible to lay them down so that they form a triangle when their ends are touching. For example, if all of the straws have a length of 6 inches. then one can easily construct an equilateral triangle using them. However, if one straw is 6 inches. long, while the other two are each only 2 inches. long, then a triangle cannot be formed. In general, if any one length is greater than or equal to the sum of the other two then the lengths cannot be used to form a triangle. Otherwise they can form a triangle.

Write a function that determines whether or not three lengths can form a triangle.

The function will take 3 parameters and return a Boolean result. In addition, write a program that reads 3 lengths from the user and demonstrates the behaviour of this function.

**Exercise 90: Does a String Represent an Integer?**

In this exercise you will write a function named isInteger that determines whether or not the characters in a string represent a valid integer. When determining if a string represents an integer you should ignore any leading or trailing white space. Once this white space is ignored, a string represents an integer if its length is at least 1 and it only contains digits, or if its first character is either + or - and the first character is followed by one or more characters, all of which are digits.

Write a main program that reads a string from the user and reports whether or not it represents an integer. Ensure that the main program will not run if the file containing your solution is imported into another program.

**Exercise 91: Is a Number Prime?**

A prime number is an integer greater than 1 that is only divisible by one and itself.

Write a function that determines whether or not its parameter is prime, returning True if it is, and False otherwise. Write a main program that reads an integer from the user and displays a message indicating whether or not it is prime. Ensure that the main program will not run if the file containing your solution is imported into another program.

**Exercise 93: Next Prime**

In this exercise you will create a function named nextPrime that finds and returns the first prime number larger than some integer, *n*. The value of *n* will be passed to the function as its only parameter. Include a main program that reads an integer from the user and displays the first prime number larger than the entered value.

**Exercise 94: Random Password**

Write a function that generates a random password. The password should have a random length of between 7 and 10 characters. Each character should be randomly selected from positions 33 to 126 in the ASCII table. Your function will not take any parameters. It will return the randomly generated password as its only result.

Display the randomly generated password in your file’s main program. Your main program should only run when your solution has not been imported into another file.

**Exercise 95: Random License Plate**

In a particular jurisdiction, older license plates consist of three letters followed by three numbers. When all of the license plates following that pattern had been used, the format was changed to four numbers followed by three letters. Write a function that generates a random license plate. Your function should have approximately equal odds of generating a sequence of characters for an old license plate or a new license plate. Write a main program that calls your function and displays the randomly generated license plate.

**Exercise 98: Hexadecimal and Decimal Digits**

Write two functions, hex2int and int2hex, that convert between hexadecimal digits (0, 1, 2, 3, 4, 5, 6, 7, 8, 9, A, B, C, D, E and F) and base 10 integers. The hex2int function is responsible for converting a string containing a single hexadecimal digit to a base 10 integer, while the int2hex function is responsible for converting an integer between 0 and 15 to a single hexadecimal digit. Each function will take the value to convert as its only parameter and return the converted value as the function’s only result. Ensure that the hex2int function works correctly for both uppercase and lowercase letters. Your functions should end the program with a

meaningful error message if an invalid parameter is provided.

**Exercise 99:Arbitrary Base Conversions**

Write a program that allows the user to convert a number from one base to another. Your program should support bases between 2 and 16 for both the input number and the result number. If the user chooses a base outside of this range then an appropriate error message should be displayed and the program should exit. Divide your program into several functions, including a function that converts from an arbitrary base to

base 10, a function that converts from base 10 to an arbitrary base, and a main program that reads the bases and input number from the user.

**Exercise 100: Days in aMonth**

Write a function that determines how many days there are in a month. Your function will take two parameters: The month as an integer between 1 and 12, and the year as a four digit integer. Ensure that your function reports the correct number of days in February for leap years. Include a main program that reads a month and year from the user and displays the number of days in that month.

**Exercise 103: Magic Dates**

A magic date is a date where the day multiplied by the month is equal to the two-digit year. For example, June 10, 1960 is a magic date because June is the sixth month, and 6 times 10 is 60, which is equal to the two-digit year. Write a function that determines whether or not a date is a magic date. Use your function to create a main program that finds and displays all of the magic dates in the 20th century.

## Arrays

**Exercise 104: Sorted Order**

Write a program that reads integers from the user and stores them in a list. Your program should continue reading values until the user enters 0. Then it should display all of the values entered by the user (except for the 0) in order from smallest to largest, with one value appearing on each line. Use either the sort method or the sorted function to sort the list.

**Exercise 105: Reverse Order**

Write a program that reads integers from the user and stores them in a list. Use 0 asa sentinel value to mark the end of the input. Once all of the values have been read your program should display them (except for the 0) in reverse order, with one value appearing on each line.

**Exercise 106:Remove Outliers**

When analysing data collected as part of a science experiment it may be desirable to remove the most extreme values before performing other calculations. Write a function that takes a list of values and an non-negative integer, n, as its parameters.

The function should create a new copy of the list with the n largest elements and the n smallest elements removed. Then it should return the new copy of the list as the function’s only result. The order of the elements in the returned list does not have to match the order of the elements in the original list.

Write a main program that demonstrates your function. Your function should read a list of numbers from the user and remove the two largest and two smallest values from it. Display the list with the outliers removed, followed by the original list. Your program should generate an appropriate error message if the user enters less than 4 values.

**Exercise 107:Avoiding Duplicates**

In this exercise, you will create a program that reads words from the user until the user enters a blank line. After the user enters a blank line your program should display each word entered by the user exactly once. The words should be displayed in the same order that they were entered. For example, if the user enters:

A screenshot of a cell phone

Description automatically generated

**Exercise 108: Negatives, Zeros and Positives**

Create a program that reads integers from the user until a blank line is entered. Once all of the integers have been read your program should display all of the negative numbers, followed by all of the zeros, followed by all of the positive numbers. Within each group the numbers should be displayed in the same order that they were entered by the user. For example, if the user enters the values 3, -4, 1, 0, -1, 0, and -2 then your program should output the values -4, -1, -2, 0, 0, 3, and 1. Your program should display each value on its own line.

**Exercise 109: List of Proper Divisors**

A proper divisor of a positive integer, *n*, is a positive integer less than *n* which divides evenly into *n*. Write a function that computes all of the proper divisors of a positive integer. The integer will be passed to the function as its only parameter. The function will return a list containing all of the proper divisors as its only result. Complete this exercise by writing a main program that demonstrates the function by reading

a value from the user and displaying the list of its proper divisors.

**Exercise 110: Perfect Numbers**

An integer, *n*, is said to be *perfect* when the sum of all of the proper divisors of *n* is equal to *n*. For example, 28 is a perfect number because its proper divisors are 1, 2, 4, 7 and 14, and 1 + 2 + 4 + 7 + 14 = 28.

Write a function that determines whether or not a positive integer is perfect. Your function will take one parameter. If that parameter is a perfect number then your function will return true. Otherwise it will return false. In addition, write a main program that uses your function to identify and display all of the perfect numbers between 1 and 10,000.

**Exercise 111: Only the Words**

In this exercise you will create a program that identifies all of the words in a string entered by the user. Begin by writing a function that takes a string of text as its only parameter. Your function should return a list of the words in the string with the punctuation marks at the edges of the words removed. The punctuation marks that you must remove include commas, periods, question marks, hyphens, apostrophes, exclamation

points, colons, and semicolons. Do not remove punctuation marks that appear in the middle of a words, such as the apostrophes used to form a contraction. For example, if your function is provided with the string "Examples of contractions include: don’t, isn’t, and wouldn’t." then your function should return the list ["Examples", "of", "contractions", "include", "don’t", "isn’t", "and", "wouldn’t"].

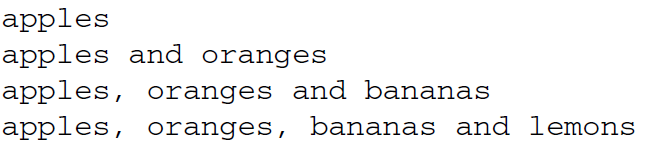
Write a main program that demonstrates your function. It should read a string from the user and display all of the words in the string with the punctuation marks removed.

**Exercise 112: Below and Above Average**

Write a program that reads numbers from the user until a blank line is entered. Your program should display the average of all of the values entered by the user. Then the program should display all of the below average values, followed by all of the average values (if any), followed by all of the above average values. An appropriate label should be displayed before each list of values.

**Exercise 113: Formatting a List**

When writing out a list of items in English, one normally separates the items with commas. In addition, the word “and” is normally included before the last item, unless the list only contains one item. Consider the following four lists:



Write a function that takes a list of strings as its only parameter. Your function should return a string that contains all of the items in the list formatted in the manner described previously as its only result. While the examples shown previously only include lists containing four elements or less, your function should behave correctly for lists of any length. Include a main program that reads several items from the user, formats them by calling your function, and then displays the result returned by the function.

**Exercise 114: Random Lottery Numbers**

In order to win the top prize in a particular lottery, one must match all 6 numbers on his or her ticket to the 6 numbers between 1 and 49 that are drawn by the lottery organizer. Write a program that generates a random selection of 6 numbers for a lottery ticket. Ensure that the 6 numbers selected do not contain any duplicates.

Display the numbers in ascending order.

**Exercise 115: Pig Latin**

Pig Latin is a language constructed by transforming English words. While the origins of the language are unknown, it is mentioned in at least two documents from the nineteenth century, suggesting that it has existed for more than 100 years. The following rules are used to translate English into Pig Latin:

• If the word begins with a consonant (including y), then all letters at the beginning of the word, up to the first vowel (excluding y), are removed and then added to the end of the word, followed by ay. For example, computer becomes omputercay and think becomes inkthay.

• If the word begins with a vowel (not including y), then way is added to the end of the word. For example, algorithm becomes algorithmway and office becomes officeway.

Write a program that reads a line of text from the user. Then your program should translate the line into Pig Latin and display the result. You may assume that the string entered by the user only contains lowercase letters and spaces.

**Exercise 116: Pig Latin Improved**

Extend your solution to Exercise 115 so that it correctly handles uppercase letters and punctuation marks such as commas, periods, question marks and exclamation marks If an English word begins with an uppercase letter then its Pig Latin representation should also begin with an uppercase letter and the uppercase letter moved to the end of the word should be changed to lowercase. For example, Computer should become Omputercay. If a word ends in a punctuation mark then the punctuation mark should remain at the end of the word after the transformation has been performed. For example, Science! should become Iencescay!.

**Exercise 118: Shuffling a Deck of Cards**

A standard deck of playing cards contains 52 cards. Each card has one of four suits along with a value. The suits are normally spades, hearts, diamonds and clubs while the values are 2 through 10, Jack, Queen, King and Ace.

Each playing card can be represented using two characters. The first character is the value of the card, with the values 2 through 9 being represented directly. The characters “T”, “J”, “Q”, “K” and “A” are used to represent the values 10, Jack, Queen, King and Ace respectively. The second character is used to represent the suit of the card. It is normally a lowercase letter: “s” for spades, “h” for hearts, “d” for diamonds and “c” for clubs. The following table provides several examples of cards and their two-character representations.

A screenshot of a cell phone

Description automatically generated

Begin by writing a function named createDeck. It will use loops to create a complete deck of cards by storing the two-character abbreviations for all 52 cards into a list. Return the list of cards as the function’s only result. Your function will not take any parameters.

Write a second function named shuffle that randomizes the order of the cards in a list. One technique that can be used to shuffle the cards is to visit each element in the list and swap it with another random element in the list. You must write your own loop for shuffling the cards. You cannot make use of Python’s built-in shuffle

function.

Use both of the functions described in the previous paragraphs to create a main program that displays a deck of cards before and after it has been shuffled. Ensure that your main program only runs when your functions have not been imported into another file.

**Exercise 119: Dealing Hands of Cards**

In many card games each player is dealt a specific number of cards after the deck has been shuffled. Write a function, deal, which takes the number of hands, the number of cards per hand, and a deck of cards as its three parameters. Your function should return a list containing all of the hands that were dealt. Each hand will be represented as a list of cards.

When dealing the hands, your function should modify the deck of cards passed to it as a parameter, removing each card from the deck as it is added to a player’s hand. When cards are dealt, it is customary to give each player a card before any player receives an additional card. Your function should follow this custom when

constructing the hands for the players.

Use your solution to Exercise 118 to help you construct a main program that creates and shuffles a deck of cards, and then deals out four hands of five cards each. Display all of the hands of cards, along with the cards remaining in the deck after the hands have been dealt.

**Exercise 120: Is a List already in Sorted Order?**

Write a function that determines whether or not a list of values is in sorted order (either ascending or descending). The function should return True if the list is already sorted. Otherwise it should return False. Write a main program that reads a list of numbers from the user and then uses your function to report whether or not the list is sorted.