Python Exercises

Exercise 1: Text Messaging

On some basic cell phones, text messages can be sent using the numeric keypad. Because each key has multiple letters associated with it, multiple key presses are needed for most letters. Pressing the number once generates the first character listed for that key. Pressing the number 2, 3, 4 or 5 times generates the second, third, fourth or fifth character.

Key	**Symbol**
1	.,?!:
2	ABC
3	DEF
4	GHI
5	JKL
6	MNO
7	PQRS
8	TUV
9	WXYZ
0	*space*

Write a program that displays the key presses needed for a message entered by the user. Construct a dictionary that maps from each letter or symbol to the key presses needed to generate it. Then use the dictionary to create and display the presses needed for the user's message. For example, if the user enters Hello, World! then your program should output:

```
4433555555666110966677755531111
```

Ensure that your program handles both uppercase and lowercase letters. Ignore any characters that aren't listed in the table above such as semicolons and parentheses.

Exercise 2: Unique Characters

Create a program that determines and displays the number of unique characters in a string entered by the user. For example, "Hello, World!" has 10 unique characters while "zzz" has only one unique character. Use a dictionary or set to solve this problem.

Exercise 3: Morse Code

Morse code[1] is an encoding scheme that uses dashes and dots to represent digits and letters. In this exercise, you will write a program that uses a dictionary to store the mapping from these symbols to Morse code. Use a period to represent a dot, and a hyphen to represent a dash. The mapping from characters to dashes and dots is shown in the following table. Your program should read a message from the user. Then it should translate all of the letters and digits in the message to Morse code, leaving a space between each sequence of dashes and dots. Your program should ignore any characters that are not listed in the previous table. The Morse code for "Hello, World!" is shown below:

.... . .-.. .-.. --- .-- --- .-. .-.. -..

Character	Code	Character	Code	Character	Code	Character	Code
A	.-	J	.---	S	…	1	.----
B	-…	K	-.-	T	-	2	..---
C	-.-.	L	.-..	U	..-	3	…--
D	-..	M	--	V	…-	4	….-
E	.	N	-.	W	.--	5	…..
F	..-.	O	---	X	-..-	6	-….
G	--.	P	.--.	Y	-.--	7	--…
H	….	Q	--.-	Z	--..	8	---..
I	..	R	.-.	0	-----	9	----.

Exercise 4: Postal Codes

The first, third and fifth characters in a Canadian postal code are letters while the second, fourth and sixth characters are digits. The province or territory in which an address resides can be determined from the first character of its postal code, as shown in the following table. No valid postal codes currently begin with D, F, I, O, Q, U, W, or Z. The second character in a postal code identifies whether the address is rural or urban. If that character is a 0 then the address is rural. Otherwise it is urban.

Province/Territory	First Character(s)
Newfoundland	A
Nova Scotia	B
Prince Edward Island	C
New Brunswick	E
Quebec	G, H, and J

[1] Morse code was originally developed in the nineteenth century for use over telegraph wires. It is still used today, more than 160 years after it was first created.

Ontario	K, L, M, N and P
Manitoba	R
Saskatchewan	S
Alberta	T
British Columbia	V
Nunavut	X
Northwest Territories	X
Yukon	Y

Create a program that reads a postal code from the user and displays the province or territory associated with it, along with whether the address is urban or rural. For example, if the user enters T2N 1N4 then your program should indicate that the postal code is for an urban address in Alberta. If the user enters X0A 1B2 then your program should indicate that the postal code is for a rural address in Nunavut or Northwest Territories. Use a dictionary to map from the first character of the postal code to the province or territory name. Display a meaningful error message if the postal code begins with an invalid character, or if the second character in the postal code is not a digit.

Exercise 5: 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 ascending order, with one value appearing on each line. Use either the sort method or the sorted function to sort the list.

Exercise 6: Reverse Order

Write a program that reads integers from the user and stores them in a list. Use 0 as a 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 7: 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 first entered.

For example, if the user enters:

first

second

first

third

second

then your program should display:

first

second

third

Exercise 8: 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 9: 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 10: Count the Elements

Python’s standard library includes a method named count that determines how many times a specific value occurs in a list. In this exercise, you will create a new function named “countRange”. It will determine and return the number of elements within a list that are greater than or equal to some minimum value and less than some maximum value. Your function will take three parameters: the list, the minimum value and the maximum value. It will return an integer result greater than or equal to 0. Include a main program that demonstrates your function for several different lists, minimum values and maximum values. Ensure that your program works correctly for both lists of integers and lists of floating-point numbers.

Exercise 11: Does a List Contain a Sub-list?

A sub-list is a list that is part of a larger list. A sub-list may be a list containing a single element, multiple elements, or even no elements at all. For example, [1], [2], [3] and [4] are all sub-lists of [1, 2, 3, 4]. The list [2, 3] is also a sub-list of [1, 2, 3, 4], but [2, 4] is not a sub-list [1, 2, 3, 4] because

the elements 2 and 4 are not adjacent in the longer list. The empty list is a sub-list of any list. As a result, [] is a sub-list of [1, 2, 3, 4]. A list is a sub-list of itself, meaning that [1, 2, 3, 4] is also a sub-list of [1, 2, 3, 4]. In this exercise you will create a function, “isSub_list”, that determines whether or not one list is a sub-list of another. Your function should take two lists, larger and smaller, as its only parameters. It should return True if and only if smaller is a sub-list of larger. Write a main program that demonstrates your function.

Exercise 12: Generate All Sub-lists of a List

Using the definition of a sub-list from Exercise 11, write a function that returns a list containing every possible sub-list of a list. For example, the sub-lists of [1, 2,3] are [], [1], [2], [3], [1, 2], [2, 3] and [1, 2, 3]. Note that your function will always return a list containing at least the empty list because the empty list is a sub-list of every list. Include a main program that demonstrate your function by displaying all of the sub-lists of several different lists.

Exercise 13: The Sieve of Eratosthenes[2]

The Sieve of Eratosthenes is a technique that was developed more than 2,000 years ago to easily find all of the prime numbers between 2 and some limit, say 100. A description of the algorithm follows:

```
Write down all of the numbers from 0 to the limit
Cross out 0 and 1 because they are not prime
Set $p$ equal to 2
While $p$ is less than the limit do
```

[2] This algorithm for finding prime numbers is not Eratosthenes’ only claim to fame. His other noteworthy accomplishments include calculating the circumference of the Earth and the tilt of the Earth’s axis. He also served as the Chief Librarian at the Library of Alexandria.

```
        Cross out all multiples of p (but not p itself)
        Set p equal to the next number in the list that is not crossed out
    Report all of the numbers that have not been crossed out as prime
```

The key to this algorithm is that it is relatively easy to cross out every *n*th number on a piece of paper. This is also an easy task for a computer - a for loop can simulate this behavior when a third parameter is provided to the range function. When a number is crossed out, we know that it is no longer prime, but it still occupies space on the piece of paper, and must still be considered when computing later prime numbers. As a result, you should **not** simulate crossing out a number by removing it from the list. Instead, you should simulate crossing out a number by replacing it with 0. Then, once the algorithm completes, all of the non-zero values in the list are prime.

Create a Python program that uses this algorithm to display all of the prime numbers between 2 and a limit entered by the user. If you implement the algorithm correctly you should be able to display all of the prime numbers less than 1,000,000 in a few seconds.

Exercise 14: 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 15: 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 kilometers) as its only parameter and returns the total fare as its only result. Write a main program that demonstrates the function.

Exercise 16: Median of Three Values[3]

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.

[3] The median value is the middle of the three values when they are sorted into ascending order. It can be found using if statements, or with a little bit of mathematical creativity.

Exercise 17: Does a String Represent an Integer?[4]

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 one 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 18: Arithmetic[5]

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 a^b

Exercise 19: Distance Between Two Points on Earth[6]

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:

$$\text{distance} = 6371.01 \times \arccos(\sin(t1) \times \sin(t2) + \cos(t1) \times \cos(t2) \times \cos(g1 - g2))$$

[4] You may find the lstrip, rstrip and/or strip methods for strings helpful when completing this exercise. Documentation for these methods is available online.
[5] You will probably find the log10 function in the math module helpful for computing the second last item in the list.
[6] Python's trigonometric functions operate in radians. As a result, you will need to convert the user's input from degrees to radians before computing the distance with the formula discussed previously. The math module contains a function named radians which converts from degrees to radians.

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 20: Area and Volume

Write a program that begins by reading a radius, r, from the user. The program will continue by computing and displaying the area of a circle with radius *r* and the volume of a sphere with radius *r*. Use the pi constant in the math module in your calculations. The area of a circle is computed using the formula $area = \pi r^2$. The volume of a sphere is computed using the formula $volume = \frac{4}{3}\pi r^3$.

Exercise 21: Area of a Triangle

The area of a triangle can be computed using the following formula, where *b* is the length of the base of the triangle, and *h* is its height:

$$area = \frac{bh}{2}$$

Write a program that allows the user to enter values for *b* and *h*. The program should then compute and display the area of a triangle with base length *b* and height *h*.

Exercise 22: Area of a Triangle (Again)

In the previous exercise you created a program that computed the area of a triangle when the length of its base and its height were known. It is also possible to compute the area of a triangle when the lengths of all three sides are known. Let *s*1, *s*2 and *s*3 be the lengths of the sides. Let *s* = *(s*1 + *s*2 + *s*3*)/*2. Then the area of the triangle can be calculated using the following formula:

$$area = \sqrt{s \times (s - s_1) \times (s - s_2) \times (s - s_3)}$$

Develop a program that reads the lengths of the sides of a triangle from the user and displays its area.

Exercise 23: Body Mass Index

Write a program that computes the body mass index (BMI) of an individual. Your program should begin by reading a height and weight from the user. Then it should use one of the following two formulas to compute the BMI before displaying it. If you read the height in inches and the weight in pounds then body mass index is computed using the following formula:

$$BMI = \frac{weight}{height \times height} \times 703$$

If you read the height in meters and the weight in kilograms then body mass index is computed using this slightly simpler formula:

$$BMI = \frac{weight}{height \times height}$$

Exercise 24: Sum of the Digits in an Integer

Develop a program that reads a four-digit integer from the user and displays the sum of its digits. For example, if the user enters 3141 then your program should display 3+1+4+1=9.

Exercise 25: 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 26: 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.

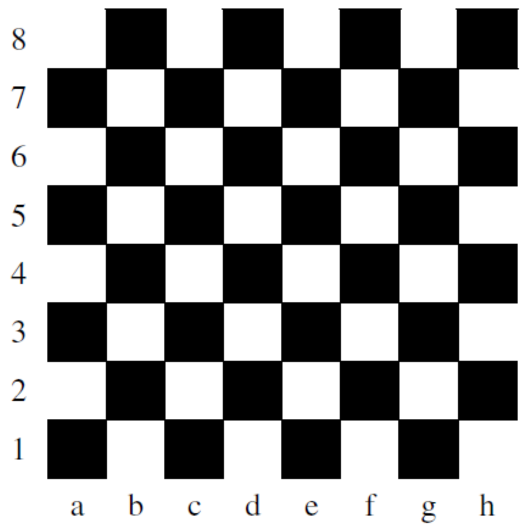
Write a program that implements the conversion from human years to dog years described in the previous paragraph. Ensure that your program works correctly for conversions of less than two human years and for conversions of two or more human years. Your program should display an appropriate error message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

Exercise 27: 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 28: 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:



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 29: Letter Grade to Grade Points

At a particular university, letter grades are mapped to grade points in the following manner:

Letter	Grade Points
A+	5.0
A	4.0
A-	3.7
B+	3.3

B	3.0
B-	2.7
C+	2.3
C	2.0
C-	1.7
D+	1.3
D	1.0
F	0

Write a program that begins by reading a letter grade from the user. Then your program should compute and display the equivalent number of grade points. Ensure that your program generates an appropriate error message if the user enters an invalid letter grade.

Exercise 30: Is It a Leap Year?

Most years have 365 days. However, the time required for the Earth to orbit the Sun is actually slightly more than that. As a result, an extra day, February 29, is included in some years to correct for this difference. Such years are referred to as leap years. The rules for determining whether or not a year is a leap year follow:

- ✓ Any year that is divisible by 400 is a leap year.
- ✓ Of the remaining years, any year that is divisible by 100 is **not** a leap year.
- ✓ Of the remaining years, any year that is divisible by 4 is a leap year.
- ✓ All other years are **not** leap years.
- ✓ Write a program that reads a year from the user and displays a message indicating
- ✓ whether or not it is a leap year.

Exercise 31: Average[7]

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.

[7] Because the 0 marks the end of the input it should **not** be included in the average.

Exercise 32: Approximate π

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

$$\pi \approx 3 + \frac{4}{2 \times 3 \times 4} - \frac{4}{4 \times 5 \times 6} + \frac{4}{6 \times 7 \times 8} - \frac{4}{8 \times 9 \times 10} + \frac{4}{10 \times 11 \times 12} - \cdots$$

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 33: 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:

```
       1   2   3   4   5   6   7   8   9  10
   1   1   2   3   4   5   6   7   8   9  10
   2   2   4   6   8  10  12  14  16  18  20
   3   3   6   9  12  15  18  21  24  27  30
   4   4   8  12  16  20  24  28  32  36  40
   5   5  10  15  20  25  30  35  40  45  50
   6   6  12  18  24  30  36  42  48  54  60
   7   7  14  21  28  35  42  49  56  63  70
   8   8  16  24  32  40  48  56  64  72  80
   9   9  18  27  36  45  54  63  72  81  90
  10  10  20  30  40  50  60  70  80  90 100
```

When completing this exercise, you will probably find it helpful to be able to print out a value without moving down to the next line. This can be accomplished by added end="" as the last argument to your print statement. For example, print("A") will display the letter A and then move down to the next line. The statement print("A", end="") will display the letter A without moving down to the next line, causing the next print statement to display its result on the same line as the letter A.

Exercise 34: Prime Factors

The prime factorization of an integer, *n*, can be determined using the following steps:

```
Initialize factor to 2
While factor is less than or equal to n do
    If n is evenly divisible by factor then
        Conclude that factor is a factor of n
        Divide n by factor using floor division
    Else
        Increase factor by 1
```

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.