Problem solving from scratch

Need we say more? Try and *hand-write* a solution to these problems in Python.

1. Variables, basic control flow, simple types, textual I/O, extending to related functions

1. State Change
At normal atmospheric pressure, water changes state to a solid at 0C or below and a gas at 100C or above. It remains a liquid at any other temperature. Write a program that will return “solid”, “liquid” or “gas” to the user depending on the temperature they enter.

2. Average Sleep Calculator
Write a program that asks you for the number of hours you slept each night for a week (7 nights), calculates the average number of hours of sleep you have had each night and then displays this result to you.

3. Age Checker (Integer in Range Checker)
Write code to read in a person’s age. Check that the value read in is between the range 0 to 120. Keep asking for an age until the value read is in the right range.

Now rewrite the answer to the previous question as a function taking a prompt for the kind of value to be read in (in the previous question, this was the age) and the upper and lower limit on that value (in the previous question this was 0 and 120). As before the function code should keep asking the user for a value until it is in the correct range, at which point the value is returned.

4. Concussion check calculator
Making use of the function you wrote in the previous question, write a program that asks the user a series of concussion related questions that require a validated whole number rating between 0 and 10 from their patient. The individual responses will then be multiplied by an importance factor and added together to generate a score between 0 and 100 that indicates how likely they are to be concussed.

The questions and their weighting for the concussion calculation are as follows

“How strongly do they feel like they’re going to be sick?”	**v_weight = 3**
“How bad is their sense of balance?”	**b_weight = 2**
“How drowsy do they feel?”	**d_weight = 1**
“How bad is their ability to remember recent events?”	**m_weight = 4**

5. Ordinal Numbers
Write out the first 31 abbreviated ordinal numbers, e.g. 1st, 2nd, 3rd, 4th, 5th, and so on. Do not simply use 31 print statements.

Now, write a function that takes a number between 1 and 31 and returns the abbreviated ordinal number for that number, e.g. 1 gives 1st, 2 gives 2nd, 3 gives 3rd, and so on.

6. Find Max without max()
Write code to read in three integers from the user and determine which of them is the largest (first, second or third one read in). You can assume that no two integers will be the same.

Now, write a function that takes three integers as parameters and returns the largest of the three, *without* using Python's max() function.

7. Guessing Game
Write a game that generates a random number between 1 and 100 and asks the player to guess the number until they get it right. Each time the player inputs their guess, the game should display a message that indicates whether the guess is too low or too high. A message should also be displayed to congratulate the player when they get it right.

8. Sum of List of Integers
Write a function that takes a single parameter containing a list of integers and returns the sum.

2. Adding in strings

9. Anagram Maker/Scrambler
Write a program which takes a single word string as user input and rearranges the letters of the word.

10. String Reversal
Write code to read in a string from the user and write it out in reverse. E.g. the string `"Programming"` would be written out as `"gnimmargorP"`. Do not use the Python shorthand for this - `s[ ::-1 ]` - instead use a loop and string manipulation.

11. Tabletop Dice Roller
Tabletop role-playing games use a lot of different kinds of dice. Their use can be written in the following format: ***`nds +/-m`*** where ***`n`*** is the number of dice, ***`s`*** indicates how many sides the dice have and ***`m`*** is a modifier which should be added or subtracted from a roll. Here is an example: **`2d6 +3`**, where two 'd-sixes' (six-sided dice) are being rolled and 3 is being added to the total.

Write a program to take an input in the above format and produce a result. The dice rolls themselves should select a random number which appears on the die.

3. Adding in lists (array lists)

12. Increment every Integer in a List
You have a list of integers. Write code to increment every value in the list by one.

13. Playlist Shuffle
Assume the following playlist is held as a list of strings, referred to by the variable `playlist`. Write a program to display a random shuffle of the playlist where all songs appear exactly once.

```
playlist = ["Ashes by Celine Dion",
            "Welcome to the Party by Diplo",
            "Nobody Speak by DJ Shadow",
            "In Your Eyes by Peter Gabriel",
            "Take on Me by a-ha",
            "If I Could Turn Back Time by Cher",
            "9 to 5 by Dolly Parton",
            "All Out Of Love by Air Supply",
            "Bangarang by Skrillex"]
```

4. Adding in dictionaries as a look-up table

14. Word Counter
Read in lines of text from the user, finishing when you receive a line with only a full-stop on it. Each line contains words separated only by spaces with no punctuation. Keep a record of how many times each word appears, and write out these counts.

15. Square Root Lookup
Use a dictionary as a lookup table to calculate and store the square roots of the first 100 positive integers. The math library contains the sqrt function – the function call `math.sqrt( 100 )` will return `10.0`. Once the lookup table is created, then repeatedly ask the user for numbers between 1 and 100 and write out the square root of the number by using the lookup table you have created. Ensure the numbers entered by the user are in range.

16. Stone, Paper, Scissors Game
Write a version of this game lasting for one round only, where the computer picks one of stone, paper, scissors randomly, and the user is invited to enter their choice. The program should then determine who has won. You should use a dictionary to represent the game

rules (paper beats stone, stone beats scissors, and scissors beat paper), as this will reduce the complexity of the code to determine the winner.

5. Adding in dictionaries used as a “record” data type

17. Older than Average
Write a program to read in the name and age for a group of people - stop reading when a full stop is entered for the name. Store each person’s data in a separate dictionary, with an entry for their name and their age, and all the dictionaries in a list. Write out the names of all those who are older than the average age.

6. Adding in file I/O

18. People over 21
The first names and ages of a group of people are stored in a file called `namesAges.txt`, one name/age pair per line, separated by a space. Write a program to output the names of all people who are 21 or over.

19. Longest Name/Reversed Names
A file `names.txt` contains first names, one per line. Write a program to:

- Write out the name that is longest, or the names, if more than one equally long.
- Write out the names in reverse order.

20. Age Lookup
The names and ages of a group of people are stored in a file called `namesAges.txt`. Write a program which repeatedly polls the user for an age and then prints out all the people of that age. If there are no people of that age, then a short message should be output to that effect.

21. Lottery Prize Calculator (Number of 3-Number Winners)
The lottery has six randomly selected winning numbers. Each number is from one to thirty nine. Participants select six numbers, and those participants selecting three of the winning numbers wins £10.

Write a program which reads in a file `selections.txt` containing a row for every participant’s attempt. An attempt consists of a unique ID number, followed by a space and then the participant’s six numbers separated by commas. The program should determine and write out the ID numbers of all the winning rows. An example file is given below:

```
123 1,2,3,4,5,6
124 1,22,23,24,25,36
125 2,22,23,27,29,32
126 2,5,7,9,23,27,29
127 3,5,7,10,30,32,34
```

```
128 4,5,7,9,24,25,37
```

22. Pretty Print Tables
You are supplied with a text file called `games.txt` containing comma separated values on multiple lines. Write a program which takes the input file and prints it in a neat table format.

Sample `games.txt` file:

```
Game,Release Date,Developer,Price
DOOM,1993,id Software,£9.99
Worms,1995,Team17,£9.99
System Shock 2,1999,Irrational Games,£12.99
Deus Ex,2000,Ion Storm,£9.99
Assassin's Creed,2007,Ubisoft Montreal,£15.99
Castlevania,1986,Konami,£9.99
The Sims,2000,Maxis,£12.99
God of War,2005,SCE,£19.99
```

Output:

```
+------------------+--------------+------------------+--------+
| Game             | Release Date | Developer        | Price  |
+------------------+--------------+------------------+--------+
| DOOM             | 1993         | id Software      | £9.99  |
| Worms            | 1995         | Team17           | £9.99  |
| System Shock 2   | 1999         | Irrational Games | £12.99 |
| Deus Ex          | 2000         | Ion Storm        | £9.99  |
| Assassin's Creed | 2007         | Ubisoft Montreal | £15.99 |
| Castlevania      | 1986         | Konami           | £9.99  |
| The Sims         | 2000         | Maxis            | £12.99 |
| God of War       | 2005         | SCE              | £19.99 |
+------------------+--------------+------------------+--------+
```

You can utilise the following function, which takes a string and an integer, and returns the original string padded out with whitespace to the length specified:

```
def whitespace(string, length):
    while len(string) < length:
        string += " "
    return string
```

23. Word Blanker
Write a program that reads in a text file called `input.txt`. The file consists of lines of words, where words are sequences of alphabetical characters separated by spaces (assume no punctuation or number chars). Your program should transform each word so that only the first letter and the last letter of the word is retained, and intermediate letters are replaced with `*` characters. The program should print out the transformed text to the terminal.

For example, suppose the `input.txt` file contains the single line:

```
A quick brown fox jumps over the lazy dog
```

The printed output will be:

```
A q***k b***n f*x j***s o**r t*e l**y d*g
```

24. Mail Merge

A mail merge requires a file containing a table of data items called `mergeData.txt` and a template mail message file called `mergeTemplate.txt`. The template contains ordinary text with special "field names" embedded; the table file contains a row of data intended for each individual message - the columns are named using the field names found in the template file. During the mail merge, a new version of the message is created, with all the special field names replaced with the actual data from a single row of the table. Hence, the `mergeData.txt` file might contain the following data:

```
name,age
Quintin,51
Derek,47
```

And the template file might look like this:

```
Hello! My name is <name> and I'm <age>.  All the best, <name>.
```

Because there are two lines of data in the data table, there'll be two messages produced as part of the operation - these are:

```
Hello! My name is Quintin and I'm 51.  All the best, Quintin.
Hello! My name is Derek and I'm 47.  All the best, Derek.
```

Note the format of the field names in the template file. You can assume that the template message contains no <> characters as part of the ordinary text.

25. Bus Timetable

A bus timetable is stored in a comma-separated file called timetable.txt. The first line of the file contains the names of the bus stops and each subsequent line contains a single service, showing the times in hh:mm format that the bus will reach each bus stop.

Write a program to read in this file. The program should then repeatedly ask the user for a time (in hh:mm format) and bus stop, and then write out the time the next bus will arrive at the given stop after the time given. You can assume that the time will always be in the right format, but note that the user may mistype the bus stop name.

Sample bus timetable.txt file:

```
Maryhill,QM-shops,Botanics,Waitrose,UniAvenue,DumbartonRd
06:30,06:35,06:40,06:43,06:46,06:52
07:00,07:05,07:10,07:13,07:16,07:22
```

```
07:30,07:35,07:40,07:43,07:46,07:52
08:00,08:05,08:10,08:13,08:16,08:22
08:30,08:35,08:40,08:43,08:46,08:52
09:00,09:05,09:10,09:13,09:16,09:22
09:30,09:35,09:40,09:43,09:46,09:52
```

And a typical interaction, using this timetable:

```
Type in the time: 08:55
Type in the bus-stop name: Waitrose
Your next service will arrive at 09:13
Type in the time: 12.17
Type in the bus-stop name: Waitrose
There are no more services at your stop today
Type in the time: 08:24
Type in the bus-stop name: CentralStation
Bus stop not found in route, please try again.
Type in the time:
```