1	Display a message such as 'Hello World!'.
2	Using a loop, display your name 10 times.
3	Ask the user for their name. Using a loop, display their name 15 times but do not advance to the next line between prints.
4	Ask the user for their name. Ask them how many times they would like it to be displayed. Using a loop, display their name this number of times.
5	Ask the user for their name. If it is the same as yours, display a message such as "Hello Bob.". Otherwise, display something like "But, you're not Bob!".
6	Validation: Ask the user to enter a number. If it is not an integer, keep asking until it is.
7	Create a dice roll subroutine that will generate a random number between 1 and 6 (inclusive). The number should be returned by the subroutine. No messages need to be displayed on the screen.

8	Modify the dice roll subroutine to accept two parameters. They should indicate the number of dice to roll and the number of sides on the dice. So DiceRoll(3, 8) would roll three, eight sided dice. The result should be returned as a list - even if only one dice was rolled.

9	Ask the user to repeatedly enter a positive number (0 is included). The program should stop when they enter a negative number. Make sure they can only enter numbers.

10	Ask the user to enter a sequence of positive integers. When they enter a negative number, stop. Display the smallest and largest number from the input provided.

11	Ask the user for a number which must be between 1 and 26 (inclusive). Validate the input - do not accept invalid responses. Using a calculation based on ASCII codes, display the letter of the alphabet that could be found at this position. For example, if the user enters 4 then you should display the letter D.

12	Ask the user for a number which must be between 32 and 126. Validate the input - do not accept invalid responses. Display the character from the ASCII character set that corresponds to this code.

13	Write a subroutine that receives a string. Validate that the string is in the format <letter><number> and only 2 characters long. The letter must be from A-C and the number must be from 1-3. Return a tuple that contains the value False if the input was invalid or the indexes for the letter and the number. For example, if the input was "B3" the tuple should be (1, 2). If the input was "bob" the tuple would be (False).

14	Write a subroutine that receives an integer between 3 and 10 (inclusive). Using ASCII characters, draw a grid on the screen that has this number of rows and columns. If the passed value is not valid your subroutine should return False but if it draws the grid it should return True.

15	Modify the subroutine from the last challenge to accept two numbers. The first is for the number of rows they would like and the second for how many columns they would like. The numbers should be defaulted to 5. Both must be numbers between 3 and 10. Draw a grid of the size entered by the user. Return False if the input is not valid or True if the grid was drawn.
16	Write a subroutine to receive a sentence as a string. Split this into a list containing strings for each word from the sentence. For example - "Hello world" -> ["Hello", "world"]. Return the list with the words in it. YOU ARE NOT ALLOWED TO USE THE BUILT-IN FUNCTION SPLIT.
17	Write a subroutine that receives a string. The string must be at least 4 characters long. Determine if this is a palindrome (reads the same forwards as it does backwards). Return True if the string is a palindrome or False if it is too short or not a palindrome.
18	Write a subroutine that will count the number of a given letter in a string. Now wrap this in a program that will ask the user to type a sentence - validate that it is at least 20 characters long. Ask them for a letter - validate that they provided a letter. Using the subroutine, count how many times the letter appears in the sentence regardless of case.
19	The isalpha and isdigit string methods determine if a string contains only letters or only numbers. Write your own isfloat subroutine that will determine if a string contains a sequence of characters that can be converted into a float. It shoudl return True or False. Now use your subroutine to determine if a string provided by the user can be converted and then do the conversion. You need to make sure it can handle negative numbers. YOU MUST NOT USE THE BUILT-IN FUNCTION FLOAT.
20	Write a subroutine to convert from Fahrenheit to Centigrade. Write another to do the opposite. F -> C: 5/9 x (F – 32). C -> F: (C x 9/5) + 32. Your subroutines will be passed a number and should return a number. Make sure they validate the data type of the input and if it is not a number return an error value of -999
21	Progressive taxation: Income tax is charged at a different rate depending on how much you earn. If you earn less than £10k you pay no income tax. If you earn less than £50k you pay 20% income tax on the money you earn that is more than £10k. If you earn more than £50k you pay 40% income tax on the income over £50k. So, if you earn £15k you will pay £1k in tax. If you earn £60k you would pay £12k in tax. Write a program that asks the user how much they earn and then calculates their annual income tax bill.

22	National Insurance: National insurance is another tax paid on your income. If you earn less than £700 per month you do not pay it. Anything you earn between £700 and £4k per month attracts a 12% national insurance charge. Anything you earn over £4k per month attracts a 2% charge. Modify your progressive tax program to also include a calculation for the amount of National Insurance that an employee would pay.

23	Pension contribution: From your salary you also pay into a pension fund arranged by your employer. In most cases this is around 7.5% of your salary. Your employer will usually top this up - typically by at least 2.5%. Calculate the total pension contribution from both the employee and the employer.

24	Payslip: Using the information you have previously calculated produce some output to explain how we get from gross pay (the amount you get paid) to net pay (the amount that gets paid to you). Include on your explination the total amount paid into your pension. The user should enter how much they get paid per month. You should use subroutines to break your code into functional units.

25	Ask the user to enter a date. Do not accept something that is not a valid date in the form DD/MM/YYYY. Check for leap years!

26	Write a subroutine to convert from binary to hex. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

27	Write a subroutine to convert from hex to binary. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

28	Write a subroutine to convert from binary to denary. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

29	Write a subroutine to convert from denary to binary. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

30	Write a subroutine to convert from denary to hex. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

31	Write a subroutine to convert from hex to denary. YOU MAY NOT USE THE BUILT-IN FUNCTIONS.

32	Write a subroutine that takes two 8 bit binary numbers and adds them together. It should return the result as an 8 bit binary number.

33	Write a subroutine that will take two values as floats - the charge for a transaction and the amount tendered. Work out the change required and return a tuple or list that contains the number of each coin required to provide the correct change. You should aim to hand over the fewest number of coins that you can.

34	Write a subroutine that accepts a number that represents a score in a test. Return the corresponding grade. Grades are calculated as E>10, D>25, C>50, B>70, A>85. A score of 10 or less is graded as U.

35	Write a subroutine that takes a camel case string with no spaces and returns a string with each word seperated by spaces. E.g. "HelloWorld" > "Hello World".

36	Write a subroutine that takes a lowercase sentence as input and returns a camel case string with no spaces.
37	Ask the user for two words. Determine if they are anagrams of each other.
38	Ask the user for two strings, strip out all punctuation (non letter characters). Also standardise the case (lower or upper) and return the new string.
39	Ask the user for two sentences, strip out all punctuation and determine if the two strings are anagrams of each other.
40	Ask the user for two sentences, strip out all punctuation and determine if the two strings are palindromes.
41	Ask the user for a number. Determine if it is plaindromic or not. If not, reverse it and add this to the original. Check for palidrome again. Repeat until you get a palindrome. Display the number you're working on for each iteration of the loop.
42	Find a needle in a haystack. Write a subroutine that receives two strings - one if the needle and the other is the haystack. Determine where the needle first occurs in the haystack. Return the position of the first letter of the needle in the haystack. If the needle doesn't exist, return -1.

43	Write a subroutine that receives a string. Remove any pairs or repeating characters. ABBA > AA > 'empty'. Do this repeatedly until the string is either empty or contains no repeating pairs of characters. Display the string on each iteration.

44	Write a subroutine to determine if a number is beautiful. A beautiful number has all of its digits such that we can extract a sequence of adjacent numbers. 123 > 1, 2, 3. 91011 > 9, 10, 11.

45	Write a subroutine that accepts a string and a number. Make the string this length by removing characters or adding spaces from/to the left of the string.

46	Produce another subroutine that will make a string a certain length by removing or adding characters to the right. Allow the user to specify the character to be used for padding.

47	Produce a subroutine that will make a string a certain length by removing or adding characters to the left and the right alternatively. This will have the effect of centering the string in a string of the provided width. Allow the user to specify the character to use for padding.

48	Ask the user for a number, find all the prime numbers up to and including the provided one.

49	Ask the user for a number, find all the factors of this number (numbers that the original can be divided by with no remainder).

50	Ask the user for their name and their score in a test. Store this information in a dictionary with their name as the key. Keep adding new entries to the dictionary until someone provides the name STOP. Provide a mechanism for the user to look up the score of someone based on their name alone. If there is no score stored for that name, say so.

51	Ask the user for their name and their score in a test. Also ask them what subject this score was for and store this in a 2D dictionary. When all the scores have been entered, ask for a name and show all the scores for that person across all subjects. You could also ask for a subject and show all the scores by individuals for that subject.

52	Write a program that asks the user for items to add to a shopping list. When the shopping list is complete, create a file and write the items to the file.

53	Write a program to read the items from a shopping list file and display them to the user.

54	Write a program that asks the user what items they have bought and remove these from a shoppin list file.

55	Create a dictionary file with words in alphabetical order. Write a series of subroutines to check if a word exists in the file, add new words to the file (maintaining the correct order) and remove words from the file.

56	Ask the user for a pair of co-ordinates. Determine how far apart they are. Now extend the program to ask the user for a set of coordinates (you work out how to stop the inputs). Find the pair of coordinates that are closest to each other and the pair that are furthest from each other. Display these pairs with the distance between them.

57	Ask the user for a nnumber between 5 and 20. Generate this number of random numbers between 2 and 50. Shuffle the numbers into two groups with the same total.

58	Morse code. Find out what the codes are for Morse Code and write a program that can decode or encode a message. Use a . For a dot and a - for a dash. A single space should be used to indicate a new letter and two spaces to indicate a new word.

59	Create a game of noughts and crosses where two players take turns. The program should determine who the winner is.

60	Update your game so that the players choose how many games they will play. Your program should keep tabs on how many games each player wins. At the end it should announce who is the overall winner and how many individual games each player won. (Deal with the possibility of a draw!)

61	Modify your game to make it possible for a player to play against the computer.

62	Write a program to generate a desck of cards and shuffle them. Ask the user how many cards they want and deal them that number.

63	Black Jack: Using your deck of cards program write a program to play Black Jack - allow the user to lpay against the computer or another human.

64	Battleships: Create a game of battleships. Use a standard board layout for the ships.
65	Battleships 2: Extend your battleships game and get the somputer to layout the ships randomly - make sure they don't overlap.
66	Battleships 3: Extend your battleships game so that we can have a 2 player game (both boards should be laid out by the computer).
67	Sudoku 1: Write a sudoku checker - make sure that a given sudoku grid is valid.
68	Sudoku 2: Write a sudoku solver - you're given a sudoku puzzle and have to solve it.
69	Sudoku 3: Create a user interface for your sudoku solver, if you haven't done so already. Use Tkinter, pygame or some other UI module.
70	Sudoku 4: Using what you've done before, generate a new sudoku puzzle and ask the user to solve it.

71	Write a program to model an ATM. You need to devise a way to store people's PIN, account number and current balance. Users authenticate themselves with the correct account number and pin combination. When they're in credit, they can withdraw or deposit money into their account. Do not let them withdraw more than they have available. Make sure you reduce the amount of money available after a successful withdrawal. Money can be depositied but the amounts must be possible with paper notes only.

72	Scrabble 1: Create a dictionary file. Lookup the number and values of tiles in a game of scrabble. Shuffle the tiles. Deal them out and ask the user to make a word. Ensure the word is possible from the tiles given. Make sure that the word is in the dictionary. Calculate the score if the word is valid and add this to the users score. Deal them more tiles. Extension: Make this a multi user game.

73	Scrabble 2: Extend your program from the last task so that the board is also represented. Display the board to the user and ask them where the word should be placed. Your program should take account of the letters that are already in place.

74	Scrabble 3: Now include all the double and tripple letter and word scores. Keep tabs on the score of the user. Extension: use a UI module to make this look like a real scrabble board.

75	Vowel square: Ask the user for the sqaure size and generate a square where each space is filled with a randomly generated letter. Display the square. Now ask the user for another square size which must be smaller than the original. Can you find any square, of the smaller size, in the original square that only contains vowels?

76	Draughts. Write a game of draughts.

77	Farm simulator: Write a simulation that has a field of a specified size. Each space can contain a seed, a plant or be empty. Every spring the seeds turn into plants. Every Autumn the plants scatter seeds. The seeds cannot land on a space that contains a plant. Each space can only contain one plant or one seed at a time. Run the simulation for 5 years showing the state of the field at the end of every spring and every autumn.

78	Farm simulator 2: Modify your simulator to include events that can happen in winter and summer. In any given winter there is a 20% chance that there will be a frost. If there is a frost there is a 30% chance that each seed will die. In summer there is a 15% chance that there will be a drought. When there is a drought each plant has a 20% chance that it can die.

79	Farm simulator 3: Modify it again. Ask the user how many years the simulation should run for, if the state of the field should be shown each season and what the chance of each event happening should be for this run.

80	Write a game of hexapawn - you'll have to investigate what that is! Allow the game to be played by two players, taking it in turn.

81	Update your hexapawn game so that a player can play against the computer.

82	Chess. Write a game of chess.

83	Text based adventure game. Write a 1980s style text based adventure game. Have a world with at least 3 worlds, 5 objects and 2 other characters that you can speak to. This is a challenge in how to represent the world and the possible commands!

84	Analogue clock: Write a program to draw an analogu clock on the screen. The clock should have three hands (hour, minute, second) and a date window. The clock should change as time goes by.

85	Particle simulator. Write a program using a framework such as PyGame to simulate some particles in an enclosed space. They should all start in random places and have random velocities.

86	Particle simulator 2. Extend the particle simulator program so that each particle has a random size and that particles collide with each other. When they collide they should bounce off each other.

87	Particle simulator 3. Extend the particle simulator again. Particles should randomly be assigned as matter or antimatter. If two particles collide they will be replaced by a single particle of their combined sizes and net velocities. If an antimatter and matter particle of the same size collide, they anhialate each other completely.

88	Write a basic calculator. It should look like a proper calculator on the screen. It only needs +, -, ÷ and x as operators but needs to work properly. The result of the previous calculation can be used for the next. Pressing the C button will reset the calculator status.

89	Add an M+, M- and MR button to your calculator.

90	Add more complex operations such as (), sin, tan, cos etc…

91	Give your calculator a programmer's mode. BIN should put the calculator into binary mode. Any number of the screen should be converted into binary. The number buttons should be limited to 0 and 1 but otherwise all other functions should operate as normal. Answers should be displayed in binary. Switching back to DEC mode will cause the binary display to be converted to denary and the other number buttons to appear / be activated.

92	Now add a HEX mode to your calculator. You'll need extra buttons for A, B, C, D, E and F. As with the binary mode, any number displayed should be converted to hexadecimal. Switching from HEX to BIN should convert the hexadecimal number to binary. It should be possible to switch from any mode to any other and get the current display shown in that number base.
93	Tank driver. Write a program using a graphical framework such as PyGame. Draw a circle which can be guided by the player by pressing the left/right keys to change direction and the up/down keys to change speed. The tank cannot go off the sides of the screen. Make it so that it is clear what the front of the tank it.
94	Tank bullet. You have a program to let you guide a tank around the screen. Now make it so that when you press the space bar a bullet is fired from the tank. The bullet should last for approx 10 seconds. It should bounce of any edges. Add obstacles to the game. The tank cannot go through the obstacles and bullets should bounce off them.
95	Tank battle. Add a second tank to your tank program. Both tanks should behave in the same way - but you must be able to tell them apart. If a tank is hit by a bullet (its own or the other tanks) then it should be destroyed. Include a health guage for each tank. When the guage gets to zero the tank loses a life. When the lives get to zero for either tank that player loses the game.
96	Tank power. Amend the tank game so that power up appear randomly on the screen. Collecting power ups give that player advantages. Examples of power ups could include: extra speed, bullets last longer, invincibility (for a limited time), teleport, stop the other player being able to fire bullets or making the other player (and/or their bullets) slow down (temporarily).
97	Animated Sprite: Find a sprite map on the internet. Write a program to show each frame of the sprite in turn. The animation should repeat.
98	Write some code to allow the user to guide an animated sprite around the screen. The sprite should change depending on the direction it is moving (you can limit it to N, E, S and W only) and should also have a resting animation. You can make the sprites yourself or get hold of a sprite map from the web.

99	PacMan. Write a PacMan clone.
100	3D flying simulation. Write a program in which the player is flying through a 3D space. They should be able to see obstacles coming towards them and should turn left or right to avoid them. The obstacles should grow as they get closer and be convincing. They can be simple geometric shapes.
101	Write a pair of programs to act as a client and a server. The server should accept connections from the client. Send a "HELLO" message to the server. The server should respond with "GOOD DAY". Display any messages that are received on the screen.
102	Modify the communication programs to allow the user at each computer to write a message to be sent to the other user. Display the messages sent and the messages received in the order they were sent on each screen.