Exercises

	This is a mix of exercises where you have to find a solution. *How you technically solve the problem is completely your decision.* *Arrays, Dictionaries, Loops, Methods, Inheritance, and so on. All known building blocks can be used.* *Do you use classes, fine. You don't use classes, also fine.* 😊 • *First, think about your problem solving technique.* • *Second, execute your solution.* • *After that, optimize your solution.*

Exercise 15.01 (Looping numbers)

Part 1

- You have a list of even positive numbers larger than 2. You can decide how long the list is. The list starts at 4, and you always add 2 to it.
- You also have a list of prime numbers.
  - When you decided to have a list till 100, you need all the primes till 100.
  - When you decided to have a list till 10.000, you need all the primes till 10.000.
- Your first task:
  - Create some code that generates those lists, collections, dictionaries, arrays. (You can choose)
  - You have already created an exercise that generates prime numbers. This is done in Exercise 09.01.

Notes

....

....

....

....

....

....

....

Notes

....

....

....

....

....

....

....

- Could you just copy and paste your solution to make it useable?

Examples of a list of even numbers

- 4, 6, …, 98, 100.
- 4, 6, 8, …, 9.998, 10.000.

Examples of a list of prime numbers

- 2, 3, 5, …, 83, 89, 97.
- 2, 3, 5, 7, …, 9.967, 9.973.

Part 2

- You have 1 object that contains even numbers till a certain value (your maximum).
- You have 1 object that contains prime numbers till a certain value (your maximum).
- Your task:
  - Check this statement:
    - Every even number can be written into a sum of 2 prime numbers, at least in one way, but there are sometimes several ways.

	Is the statement true? *Can you find one even number that is not writeable as a sum of 2 primes?*

Examples of the check

- 4 = 2+2 (only one way).
- 6 = 3+3 (only one way).
- 8 = 3+5 (only one way).
- 10 = 3+7 = 5+5 (two ways).
- 100 = 3+97 = 11+89 = 17+83 = 29+71 = 41+59 = 47+53 (six ways).

Exercise 15.02 (Looping employees)

	The exercise below is a difficult one. *But it is a concept or pattern that happens a lot.* *First create a battleplan, and then execute.*

Part 1

- In a company, you have a list of employees.
- From this list of employees, there is one main responsible. This person / employee does not have a boss, because he / she is the chief of the company.
- All other employees must have a responsible.
- Employees that are responsible, have a list in them with the employee numbers of persons that they are directly responsible of, this can go only 1 level deep.
- Your task:
  - Make classes and class diagram that makes this possible.
  - Make a test program where you are able to create the test data given below in the example and where you can show who is responsible of whom.
  - You start with creating the boss (top level in the company).
  - When you add other employees:
    - Make sure that you know the boss of the person.
    - Have the correct list of employees that the boss is responsible of.
  - How to test the result.
    - You should be able to show who is the boss of somebody.
    - You should be able to show the employees somebody is responsible of.

Notes

....

....

....

....

....

....

....

Example of a list of employees

EmployeeNumber	Name	FirstName	Responsible
1	Pfeiffer	Michele	
2	Jolie	Angelina	1
3	Pacino	Al	1
4	De Niro	Robert	3

- Michele is responsible of Angelina (first level) and Al (first level).
- Angelina is responsible of nobody.
- Al is responsible of Robert (first level).
- Robert is responsible of nobody.
- The boss of Angelina is Michele.
- The boss of Michele is nobody.

Part 2

- We are going to add some data quality checks.
- For data consistency:
    - It is not be possible to enter a second person that has no boss.
        - There is only one person that can be the chief.
        - Check this in code, when you change the value of a responsible into nothing.
        - Check this in code, when you add or change a new top chief.

Example of a list of employees that should go wrong

- Adding Angelina should go wrong, because there must be a responsible.

EmployeeNumber	Name	FirstName	Responsible
1	Pfeiffer	Michele	
2	Jolie	Angelina	

Notes

Part 3

- We are going to add some data quality checks.
- For data consistency:
    - It is not be possible to give a person a boss that does not exist.
        - So the number of the Responsible should exists in the list of employees.
        - Check this in code, when you add or change the value of the responsible.

Example of a list of employees that should go wrong

- Adding the Al should go wrong, because there must be a responsible that exists, and employee number 4 does not exist.

EmployeeNumber	Name	FirstName	Responsible
1	Pfeiffer	Michele	
2	Jolie	Angelina	1
3	Pacino	Al	4 (this is wrong)

Part 4

- We are going to add some data quality checks.
- For data consistency:
    - It is not be possible to create a circular reference of bosses.
        - An employee can't be a boss of somebody that is responsible for you.
        - This can go multiple levels deep.

Notes

....

....

....

....

....

....

....

	There are several solutions possible. *Tip: You can work with a list of employees where the boss is responsible of, but more than one level deep.* *Second Tip: You drill down into the hierarchy of bosses. (Recursive solution).*

First example of a list of employees that should go wrong

- Adding the employees below should work.
- Changing the boss for Angelina into 3 (Al) should fail.
  - This is not allowed.
    - Angelina is boss of Al, and Al is boss of Angela.

EmployeeNumber	Name	FirstName	Responsible
1	Pfeiffer	Michele	
2	Jolie	Angelina	1
3	Pacino	Al	2

Examples of a list of employees that should go wrong

- Adding the employees below should work.
- Changing the boss for Angelina into 4 (Robert) should fail.
  - This is not allowed.
    - Angelina is boss of Al, and Al is boss of Robert, and Robert is boss of Angelina.

EmployeeNumber	Name	FirstName	Responsible
1	Pfeiffer	Michele	
2	Jolie	Angelina	1
3	Pacino	Al	2
4	De Niro	Robert	3

Notes

Notes

Exercise 15.03 (Huffman data compression)

On internet you find a lot of websites with explanation and coding in C# to implement this algorithm.

The goal is that you make it by yourself, but a good habit is to learn from others. Look for inspiration. Look for solution techniques. Some techniques are not explained yet in the classroom, so if you don't recognize the pattern, look for something else.

The exercise is complex, due to the many things that must be done. And they have to work together. There is also a recursive element in it.

Part 1 – What do we want to do?

Assume you have a file or string variable with the content "taartstraat". When this document or variable is saved in ASCII-code, you have this numbers.

Character	Ascii-code	Binary code	Hex code
t	116	01110100	74
a	97	01100001	61
r	114	01110010	72
s	115	01110011	73

- Character set = "taartstraat" (length = 11 characters).
- Binary code (spaces are for readability) = 01110100 01100001 01100001 01110010 01110100 01110011 01110100 01110010 01100001 01100001 01110100 (length = 88 bits).
- Hexadecimal code (spaces are for readability) = 74 61 61 72 74 73 74 72 61 61 74 (length = 88 bits).

We need a system to convert this binary set to a shorter one, and also a system to convert it back into a readable format.

This system (and others) is used in commercial compress tools like WinZip, 7Zip, …

Part 2 – A proposal that is not useable (just for explanation)

Characters	Ascii-code	Binary code	Hex code	Proposal
t	116	01110100	74	0
a	97	01100001	61	1
r	114	01110010	72	01
s	115	01110011	73	10

- Character set = taartstraat (length = 11 characters).
- Compress code = 01101010001110 (length = 14 bits).

The problem here is that we can't go back. Because the code "01" can be translated into "r" or "ta".

So we need to find another system.

The Huffman code algorithm will help us.

Notes

Notes

Part 3 – The Huffman code algorithm

Step 1.

Count how many times every character is used in the text “taartstraat” to convert.

Characters	Count
a	4
t	4
r	2
s	1

Step 2.

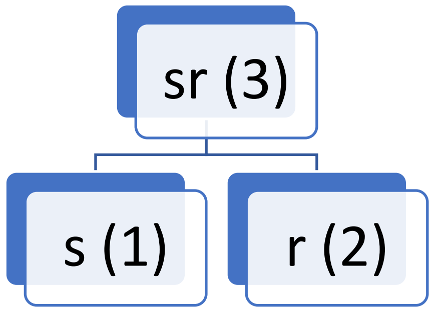
Sort this list from low to high. “a” en “t” can be switched, if so, you have another correct solution.

Characters	Count
S	1
R	2
a	4
t	4

Step 3.

We take the first 2 items (if there are two) and combine those characters and also sum the count. And this combination is placed back in the list in the correct order (using the sum of the count).

The list must always be sorted from low to high on the value of Count.



Notes

Step 4.

We also make a tree (or a node of a tree) with that information. (See the picture beside). The top node is the combination, 2 child nodes are the items that were used to create this top node.

Characters	Count
s	1
r	2

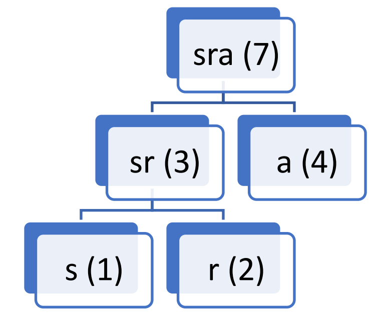
Becomes:

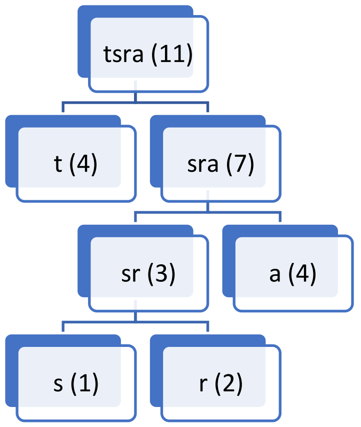
Characters	Count
sr	3

Step 5.

And placed back in the list at the correct place, sorted by Count.

Characters	Count
sr	3
a	4
t	4





Step 6.

Repeat step 3 till 5 until there is only one row left.

Characters	Count
sr	3
a	4
t	4

Becomes.

Characters	Count
t	4
sra	7

Becomes.

Characters	Count
tsra	11

Notes

....

....

....

....

....

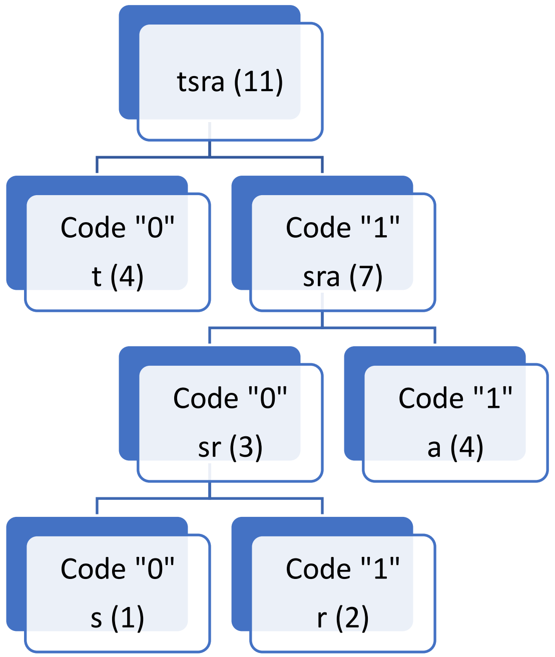
....

....

Step 7.

The tree that is result of this routine, will now be used to create some codes.

You start at the top node, and every child to the left is a zero (0) and every child to the right is a one (1).



This structure is used to create unique codes for every letter in the list. Just go from the top to the letter you want to code.

Characters	Ascii-code	Binary code	Hex code	Huffman code
t	116	01110100	74	0
a	97	01100001	61	11
r	114	01110010	72	101
s	115	01110011	73	100

Notes

Notes

....

....

....

....

....

....

....

This table is used in 2 directions. To compress the original text and decompress the result of the Huffman Code toward the original text.

- Character set = taartstraat (length = 11 characters).
- Compress code = 011111010100010111110 (length = 21 bits).

	As exercise you can do the decryption and you will see that there is no mistake possible.

Part 4 – Program this

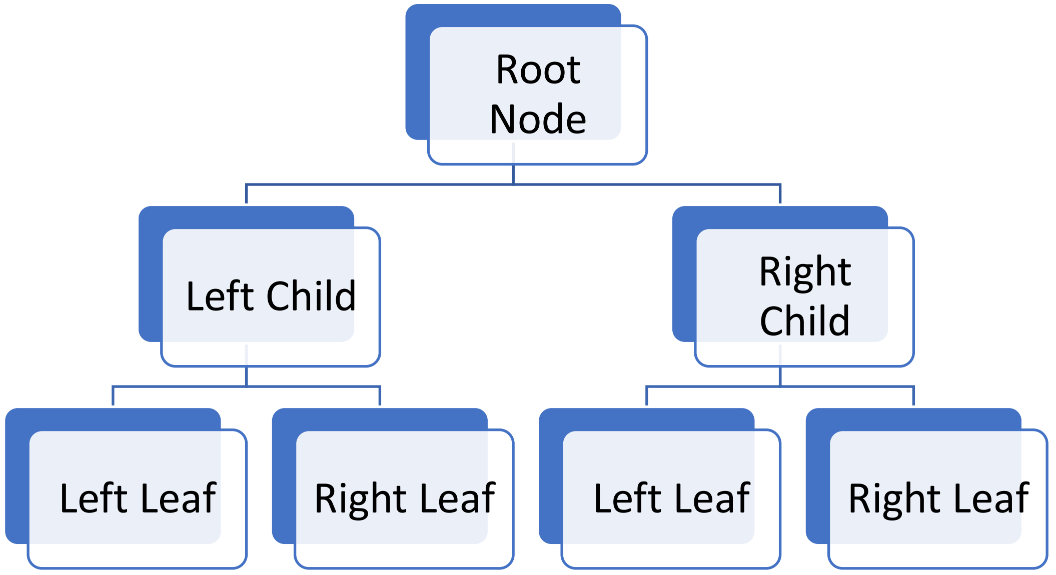
	There are a lot of possible implementations. The general idea is to have several classes to work together. *Below some guidelines with your solution.* *Start small and make sure that something you have created is tested and working fine.* *Small steps is a very good advice.* *When you copy things from the internet, make sure you understand what the code does. Make also sure that it works, because I have found several examples that are incorrect.*

Guideline 1

- Having a list of things is a good starting point. That collection needs to be sortable. In exercise 14.02 you have done something similar with an array of texts.
- You will need to add some things to that list, you will need to sort it, and then take the 2 top items (if there are 2 items) to do something with it. I personally think that a queue is a good type for the list, but it can be done with other types like array or just a normal list.
- It is also possible to work with several lists at the same time.

Guideline 2

- You will need an object that is your tree. This tree consists of nodes, so an object node is also necessary.
- Both (tree and node) are best defined as classes with methods.
- There is a root node, the top of the tree.
- A node can be parent of a right child.
- A node can be parent of a left child.
- A node that has no child is generally called a leaf.
- A branch is from the top node till a leaf.



Notes

Notes

....

....

....

....

....

....

....

Guideline 3

- Your actual test program must be able to do several things.
- Ask a text.
- Use the Compress functionality to zip it. Text to smaller sized text. This is the zipped result.
- Use the Unzip / Decompress functionality. The zipped, compressed text, must be unzippable, so you can go back to the original text.
- When the original text is returned back after compress (zip) and decompress (unzip), it usually means you have a good implementation.
- A good practice is that you also have a kind of visualisation to what code every character is zipped (the Huffman codes that are generated).

	I hope you have fun with this exercise. *It is perfectly possible that your program gives a result, but that your result is different from the manual exercise given. This is not a problem. The reason is, how is it sorted and what do you put as left node or right node.* *There is no unique solution to the problem.*

Other possible solution

Characters	Ascii-code	Binary code	Hex code	Huffman code
t	116	01110100	74	10
a	97	01100001	61	0
r	114	01110010	72	111
s	115	01110011	73	110

- Character set = taartstraat (length = 11 characters).
- Compress code = 100011110110101110010 (length = 21 bits).