Zoea

Zoea Visual Examples

Version 1.0

Table of Contents

1. Introduction

Zoea is a simple declarative programming language based on the concept of inductive programming. Instead of writing code in the conventional sense the Zoea developer describes the behaviour of a program as a set of example scenarios consisting of inputs, derived values and outputs. As a result Zoea programs resemble a set of functional test cases. The Zoea compiler uses AI to turn these test cases into executable code. Zoea programs can also be combined to form larger programs of any size.

Zoea Visual is a visual programming language. A visual language allows a user to produce software using a graphical notation - typically diagrams - instead of writing code. Zoea Visual is built on top of the Zoea language and being visual it is easier to learn and use. It is also more powerful as it extends Zoea with a number of new features including data flow dependencies and subsidiary test cases. As a result Zoea Visual can be used to build larger programs quickly and easily. Visit https://zoea.co.uk/ for more information on the Zoea and Zoea Visual languages.

This document presents a set of simple example programs in the Zoea Visual programming language. The tasks that the examples implement mainly come from the Rosetta Code web site [1]. In some cases the tasks and/or task descriptions have been modified slightly to make them easier to understand. Each task is briefly described and a complete Zoea Visual code solution is provided. A discussion of the approach and implementation details is also included.

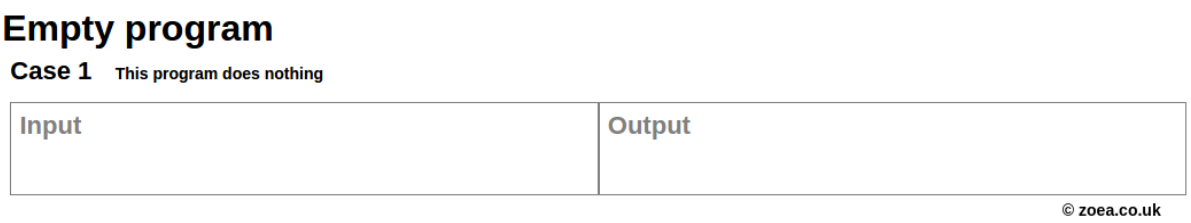
2. Examples

2.1 Empty Program

Task

Create the simplest possible program that is still syntactically correct.

Solution



Discussion

This is a valid Zoea Visual program that has no input and no output. It does however have a case comment indicating that it does nothing.

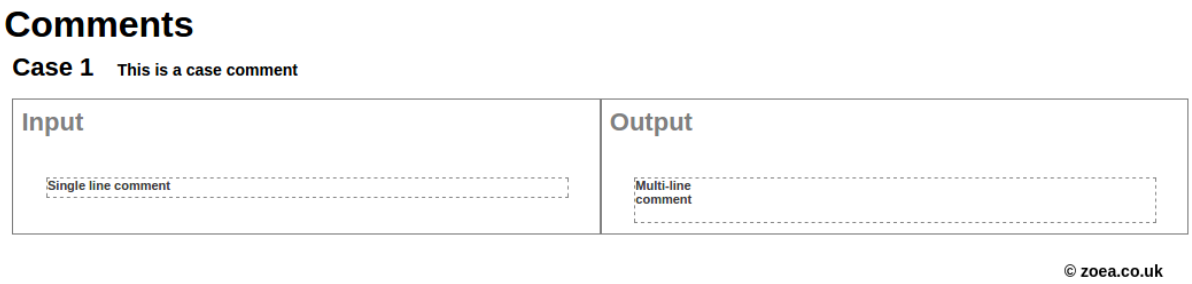
All of the examples in this document were produced using the Zoea Visual editor [2] and are the listing view of their respective programs. The listing view is similar to the Zoea Visual cases screen but does not include any of the menus or buttons. It also has a slightly different layout.

2.2 Comments

Task

Show all ways to include text in a language source file that's completely ignored by the compiler or interpreter.

Solution



Discussion

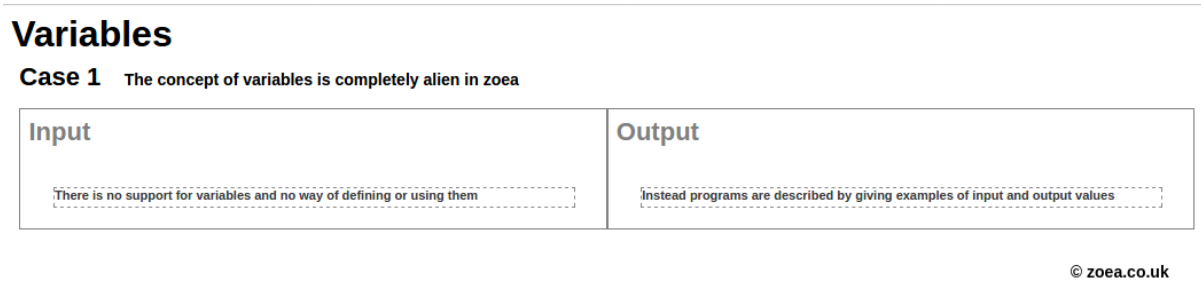
A comment can be provided for each case in the comment field on the case editor screen. This is displayed beside the case number on the listing view. In addition any number of comments can be included in any column. In Zoea Visual comments are represented by a box with a dashed border. Single and multi-line comments use the same component. Comments can include any amount of text and the comment box grows and shrinks automatically to fit its content. Comments larger than 50% of screen height will however require scrolling.

2.3 Variables

Task

Demonstrate a language's methods of variable declaration, initialization, assignment, data types, scope, referencing and other variable related facilities.

Solution



Discussion

There are no variables of any kind in Zoea or Zoea Visual. This complete avoidance of variables was deliberate. Much of the complexity associated with learning and using conventional programming languages comes from the need to name, remember and manage state. For example, when a developer uses another language to write a program they need to keep in mind the values of all variables at every point in the program. In effect they need to have a mental virtual machine that can execute their program and also store and access the results as they change during execution. With Zoea Visual the internal state of the program data is always completely visible.

There were times during the development of Zoea where it would have been easier from a language implementation point of view to allow at least some variables but this was considered to be a slippery slope. The simplicity of Zoea is more a result of what is not included rather than that which is. Also, the primary focus of the language is the value of immutable static data elements rather than the identity of the containers in which data is stored.

2.4 Hello World

Task

Display the string "Hello world!" on a text console.

Solution

Discussion

Having no input or reference data in a program means there can be no derived values or corresponding computation. As a result any output value will always be the same.

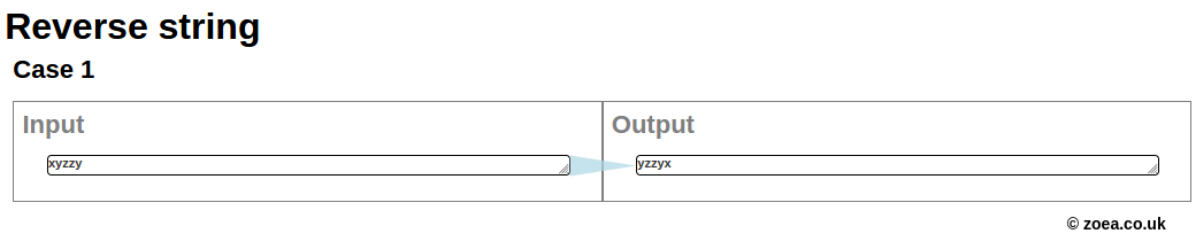
It would also be possible to add any number of additional output data elements. These would also simply be output each time the program is run.

2.5 Reverse String

Task

Take a string and reverse it. For example, "asdf" becomes "fdsa".

Solution



Discussion

This example has both an input and an output. The blue triangle connecting them is a dependency. This means that the input value is used in some way to produce the output value. Dependencies are optional but should always be included as they make the developer intention more explicit and also let the compiler operate much more quickly. Larger Zoea Visual programs would take a long time to compile without dependencies.

As with all Zoea Visual programs it should be fairly obvious to a human reader what data transformation occurs at each step. If this is not the case then it is a good idea either to add another derived value or to include a comment.

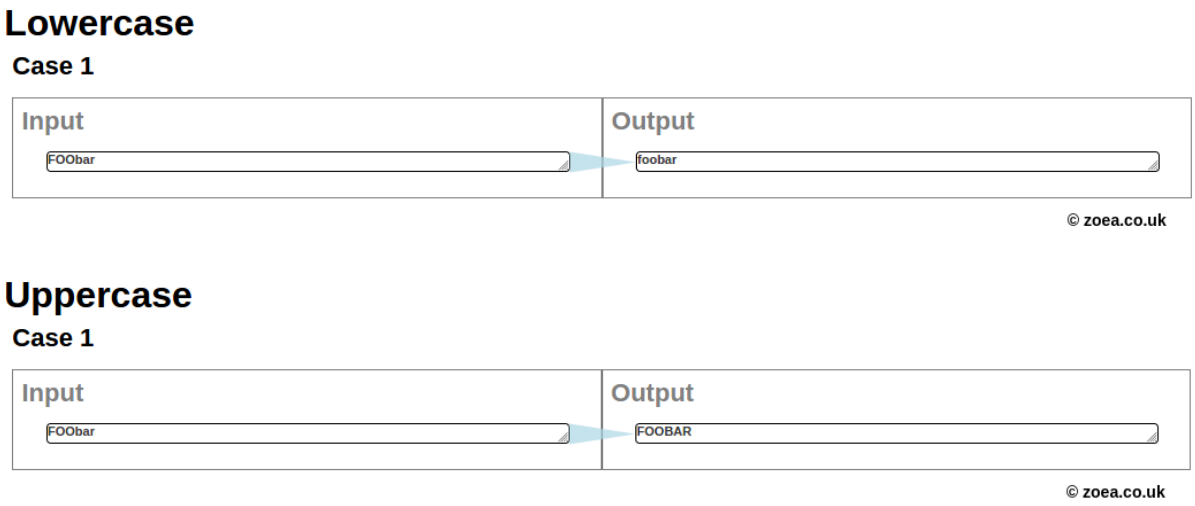
Many other possible test cases could have been included. These include single character, empty string and numeric inputs. Zoea Visual treats numbers and strings interchangeably. All numbers can be treated as strings and any string that looks like a number can also act like one.

2.6 String Case

Task

Convert a mixed case string to lower and upper case.

Solution



Discussion

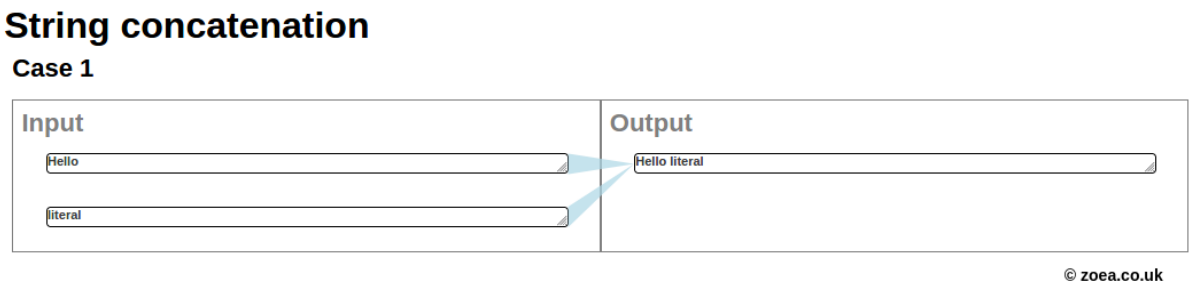
Note that we have two different programs here rather than two test cases for the same program. Both of these examples would also work with a single letter inputs and outputs but the use of multiple letters and mixed case helps to avoid any potential doubt about what is intended.

2.7 String Concatenation

Task

Concatenate two string variables.

Solution



Discussion

This program has two inputs and there are dependencies from each input to the output. This means that the values of both inputs are somehow involved in producing the value of the output.

When we look at a Zoea Visual program we don't often start on the left and follow the arrows to the right. Instead it is more useful to focus on one target element at a time together with all of its source dependencies. That way we can consider the role that each of the source elements plays in creating the target value. When we understand how one target value is produced we can move on to the next.

Note that the test case also inserts a space character between the concatenated strings. Zoea is able to determine that this space doesn't come from any of the inputs so it creates an additional internal string value as part of the generated program.

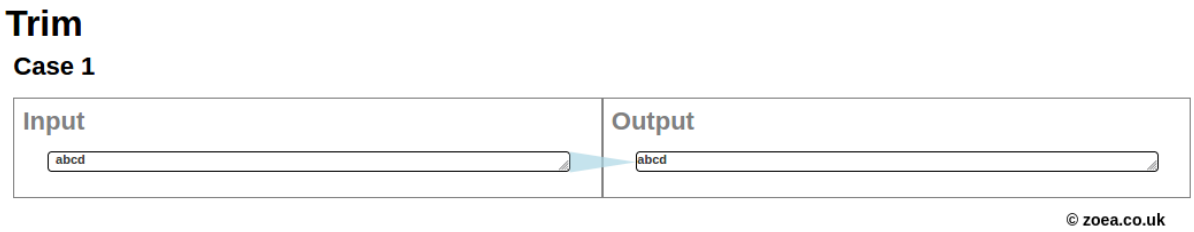
Any fixed text that is required in the output can simply be included in output data elements. It can also be specified as a data column value and linked to the appropriate output using a dependency. Zoea Visual also has a label data element that can be used to produce fixed output. Labels can occur in any column and are otherwise ignored by the compiler.

2.8 Strip White-space From A String

Task

Demonstrate how to strip leading and trailing white-space from a string.

Solution



Discussion

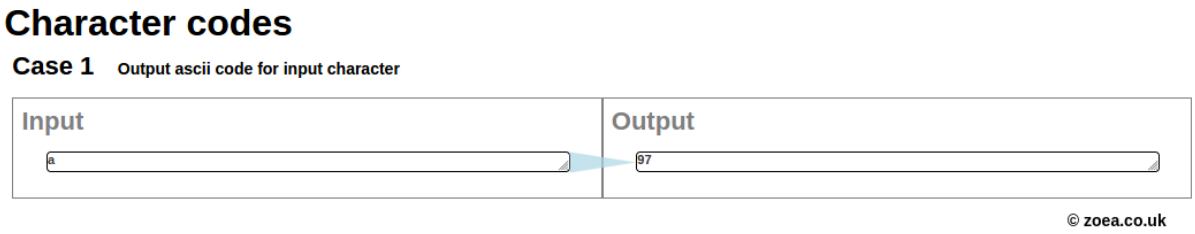
The input in this example has leading and trailing space characters while the output doesn't. Removing spaces in this way is a fairly common activity in computer programs and most programming languages have a specific function for this purpose. If the spaces in the input had been a different character such as 'x' then the intent of the test case would be less obvious. We might have meant delete all the 'x's, delete the first and last character, return the middle four characters or many other alternatives. In such cases we need to provide multiple test cases rather than just one.

2.9 Character Codes

Task

Given a character value in your language, print its ASCII code.

Solution



Discussion

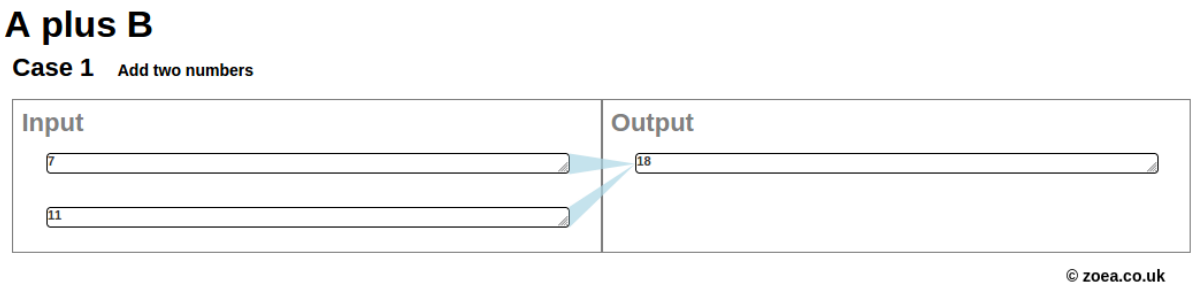
To specify this program you have to know or find out what the ASCII code for at least one character is. It doesn't make any functional difference which character is chosen but it is easier for the user to see the input if it is a non-white-space character. It is not necessary to add any more test cases as there is no simpler way to get from 'a' to '97'.

2.10 A plus B

Task

Add two input integers and display the result.

Solution



Discussion

In order to choose the best test data for a numeric operation it is useful to understand how Zoea evaluates potential solutions. The most important thing to remember is that Zoea will select the simplest solution from all the candidates that it discovers. Often there can be many plausible solutions for a given numeric test case. For example the test case [7,11] → 18 could correspond to any of the following solution candidates:

- input(1) + input(2);
- input(1) + 11;
- input(1) * 2.5714285714285716;
- input(2) + 7;
- input(2) * 1.6363636363636365;
- 25 – input(1);
- and so on.

In order to choose the simplest Zoea assigns each candidate solution a complexity score which is based on the number of operators and operands the term. So input(1)+input(2) has a base complexity score of 3 – one each for the two operands and one for the operator. This score is also weighted such that any additional constants in the term will increase the score. For example, input(1)+11 still has a base complexity score of 3 but its weighted complexity score is 4 as we add one for the integer constant (11). Floating point numbers also have a higher weight than integers. So input(1)*2.5 would have a weighted complexity score of 5.

This approach reflects the heuristic that solutions composed from available inputs are preferable to those where additional values have to be conjured up from nowhere. Also if we do have to introduce additional values then the fewer required and the simpler they are the better. Solutions involving string operations are also evaluated in a similar way.

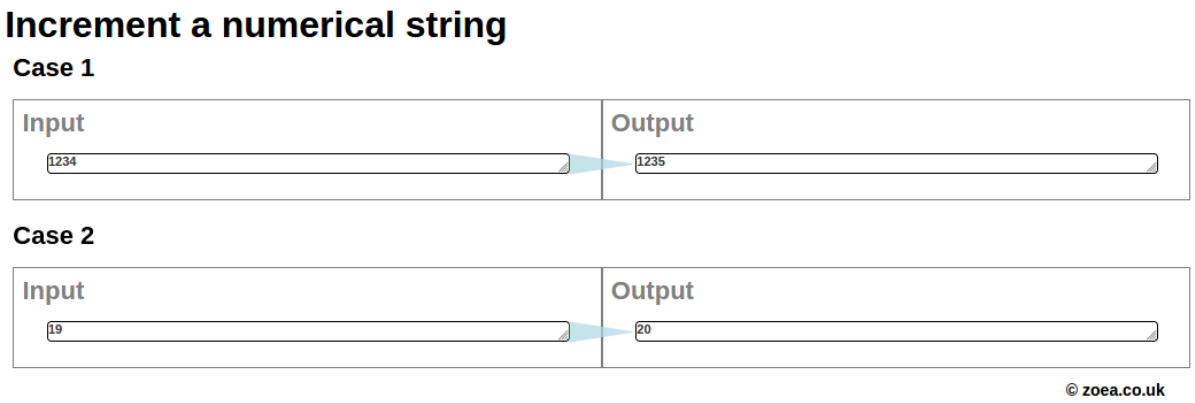
Understanding this approach makes it easier to come up with suitable test case values. The only other thing we need to watch out for are test cases with potentially ambiguous interpretations such as [2,2] → 4. With a single test case this could just as well be 2+2 or 2*2 as the weighted complexity score does not rank operators.

2.11 Increment a Numerical String

Task

Increment a numerical string.

Solution



Discussion

The problem is slightly (but not much) harder than the case where we add two numbers. This is because where we add two numbers we already have all of the data required to form the solution. In this case we also have to create an unknown value that was not provided as an input.

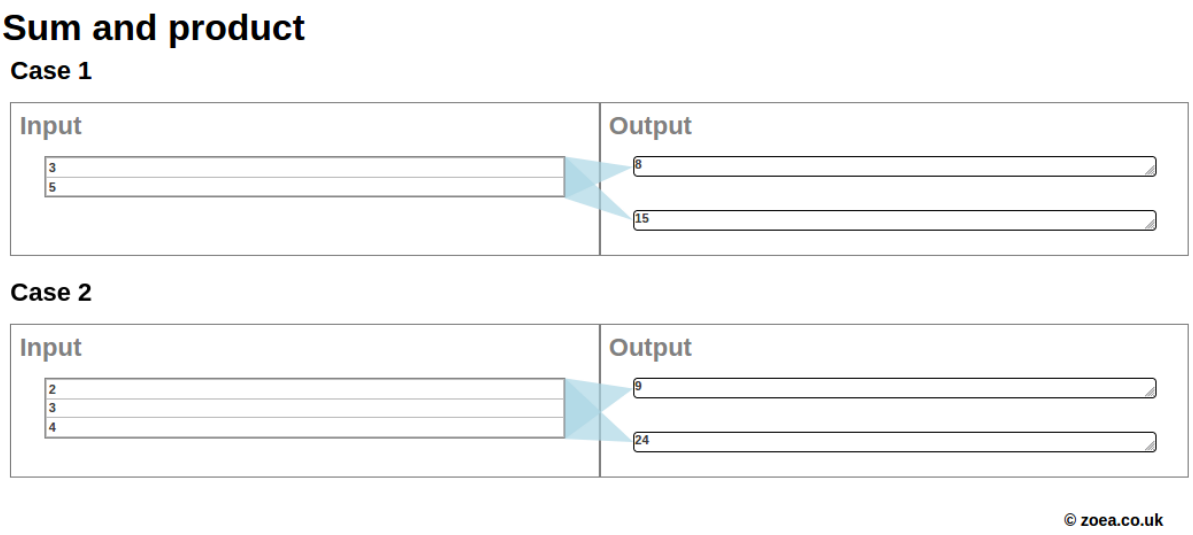
We have used two test cases in the solution. This task also works if just the first or second test cases are used alone. Using one test case for a problem like this runs a slight risk of Zoea misinterpreting the transformation as a string operation. For example case 1 in isolation could instead be interpreted as meaning replace the last character in the input with '6'. This would only be a problem if the numeric solution was more complex than the string operation which in this case it isn't.

2.12 Sum and Product

Task

Compute the sum and product of an array of integers.

Solution



Discussion

So far we have only seen examples with single value inputs and outputs. This example uses a list as input and also has two outputs. A Zoea Visual list is equivalent to a one dimensional array in other programming languages. Notice that the size of the input list is different in the two test cases. This together with the fact that we have used a three element list in case 2 reinforces idea that we are working on the entire list rather than particular elements within it.

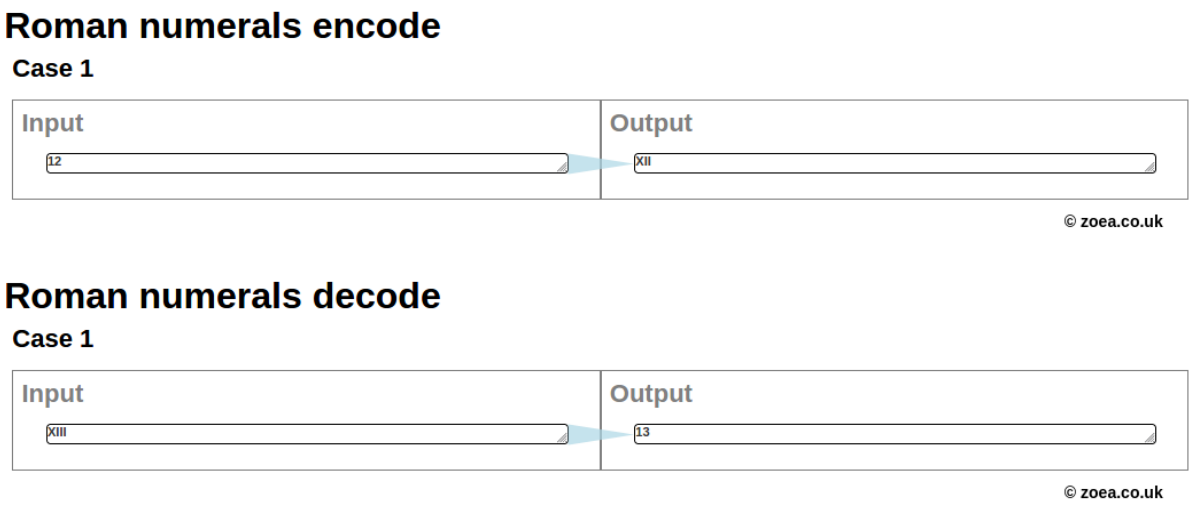
When we look at a Zoea Visual program with multiple output values such as this it is useful to think of it as really being the equivalent of multiple superimposed sets of test cases. There is one set of test cases for the first output value (input1 → 8; input2 → 9) and another set for the second output value (input1 → 15; input2 → 24). In other words the code that produces the output for the real test cases is a composite of the code that produces the first output and the code that produces the second output.

2.13 Roman Numerals

Task

Convert decimal integers to Roman numerals and back again.

Solution



Discussion

Conversion of Roman numerals as well as a long list of other things are built into Zoea rather than relying on the user or third party libraries to provide them. This is because we want to integrate what we could call 'general knowledge' into the Zoea reasoning process. It would of course be possible for the user to build their own Roman numeral conversion software using Zoea if they wanted to. However, Zoea is trying to move beyond the model of where the user has to spell out everything for the computer. So if we see a Roman numeral in a date, address or the name of a publication we expect to treat it like any other rendering of a number.

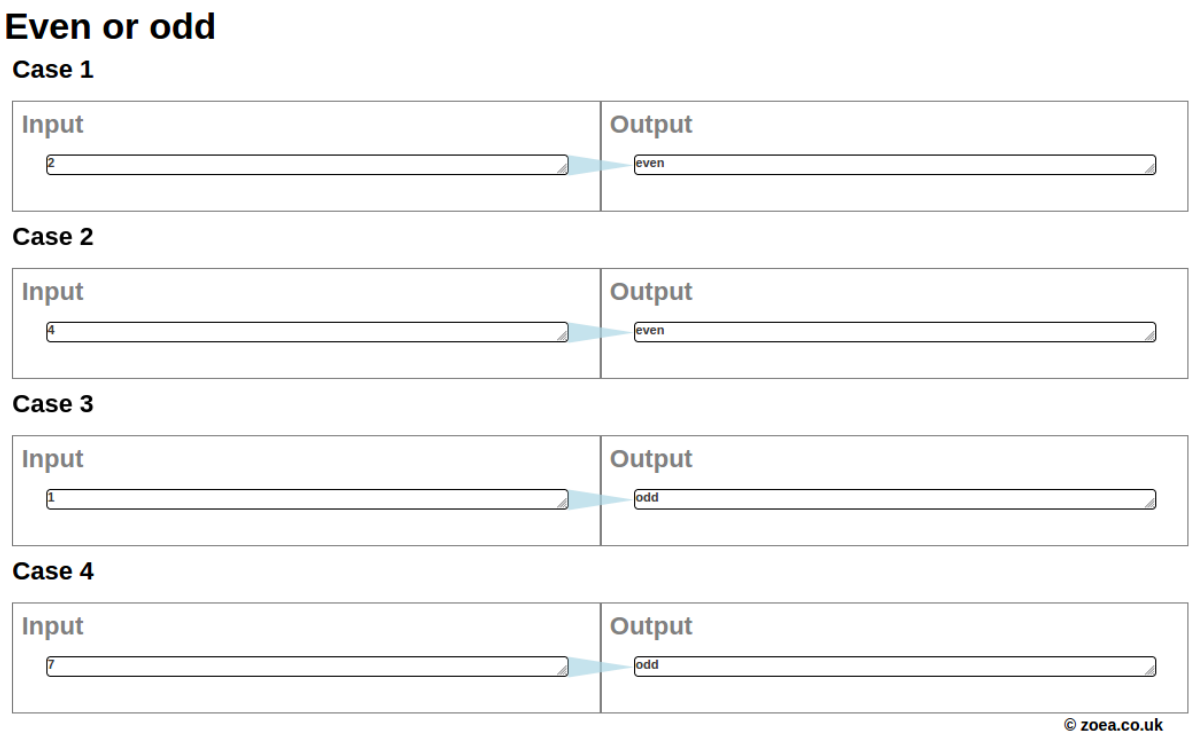
When Zoea sees Roman numerals or other things that it recognises in test cases, one of the things that it does is create a new set of synthetic test cases where those objects are converted to their canonical representation. Roman numerals are converted internally to decimal numbers. Zoea then tries to come up with a solution for the synthetic test cases. This effectively means that Zoea can produce programs that include any form of numerical operation using any combination of representations of numbers. The same principle also applies to strings and their various representations and encodings.

2.14 Even or Odd

Task

Test whether an integer is even or odd.

Solution



Discussion

This task is a simple example of a conditional statement. Zoea generates conditionals when it is unable to identify a single solution that works across all test cases. When a test case is compiled Zoea identifies every solution that works for that test case – we call that a case solution. Each case solution that is found is immediately evaluated across all test cases to see if it works for them as well. In the end we have a list of case solutions and for each case solution we have a list of test cases for which that solution is applicable. We then create sets of case solutions where the applicable list for each set covers all of the test cases – we call these solution sets. This starts with the case solutions with the largest applicable lists so we tend to generate the smallest possible solution sets early in the process. If a single case solution covers all of the test cases then we don't need a conditional. Otherwise we take the solution sets with the smallest number of case solutions and determine what the conditions are for each case solution to be applicable. Basically this compares the input and derived values of the relevant cases together with a set of functions that we call discriminators. These are combined with standard logical operators to produce a set of conditional expression hypotheses. The simplest combination of logical expressions is found by creating synthetic boolean test cases and using the same case solution/solution set approach recursively. The simplest single solution that covers all of the synthetic cases is the logical expression that we are looking for.

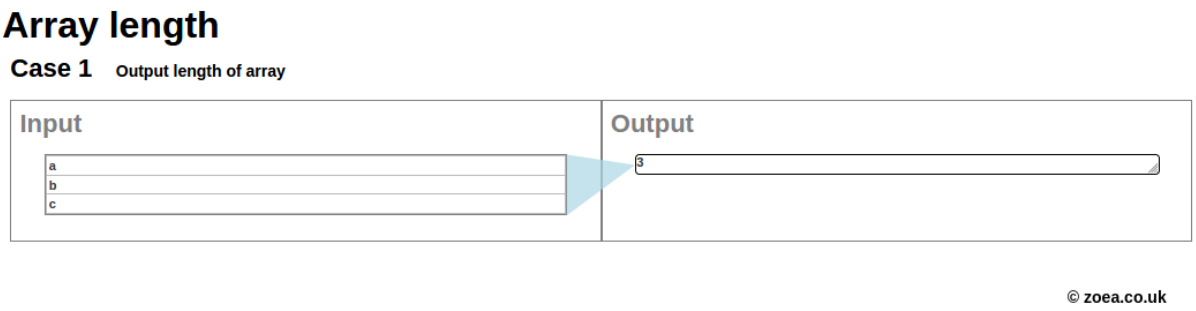
This task can also be solved with just the first three test cases. If only two cases were used then the conditional would likely be expressed in terms of the literal input values (e.g. input==X or input<X) which are not what is required.

2.15 Array Length

Task

Determine the number of elements in an array.

Solution



Discussion

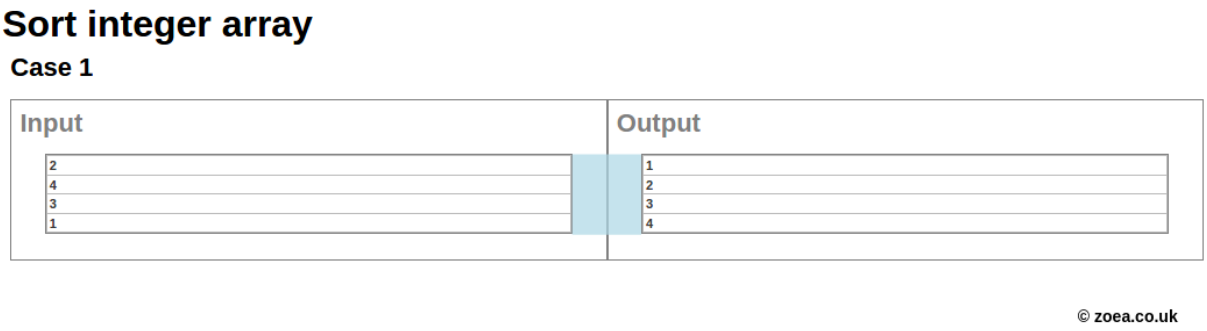
This is a simple problem and as there is no conditional logic it is easily solved with a single test case. As always the selection of the test data is important. Numeric input values have been deliberately avoided so as to reduce the risk of any mathematical derivation. We have also avoided input elements with a string length of three or indeed the digit '3' as a substring. The input as small as possible which is always good practise with Zoea.

2.16 Sort an Integer Array

Task

Sort an array (or list) of integers in ascending numerical order.

Solution



Discussion

Both the input and output elements in this example are lists. This means that the shape of the dependency is a little different. Dependencies that target single values end in a point whereas those that target composite elements (lists, tables, table rows and objects) cover the complete left hand side of the target. This makes it more obvious for the viewer that the result is a composite rather than a single value.

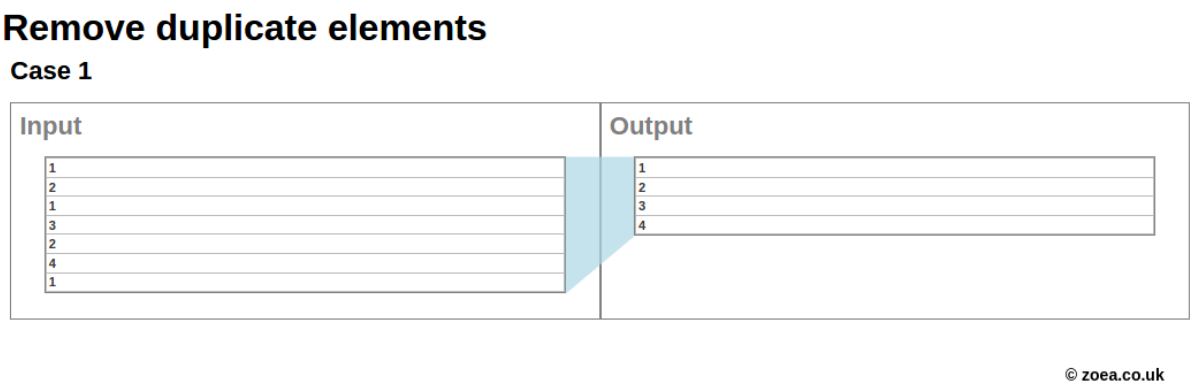
The choice of the order of the input list values is also significant. If instead the input had been [4,3,2,1] then the operation would be recognised as reverse rather than sort. Zoea can also determine complicated multi-column sort orders on tables as well as selection and filtering criteria if required.

2.17 Remove Duplicate Elements

Task

Given an Array, derive a sequence of elements in which all duplicates are removed.

Solution



Discussion

The test case has been formulated to reflect the problem where duplicates can occur anywhere later in the list rather than simply runs of repeated elements. Note also that the output numbers are in the same order as they first occur in the input – they only happen to be in sorted order.

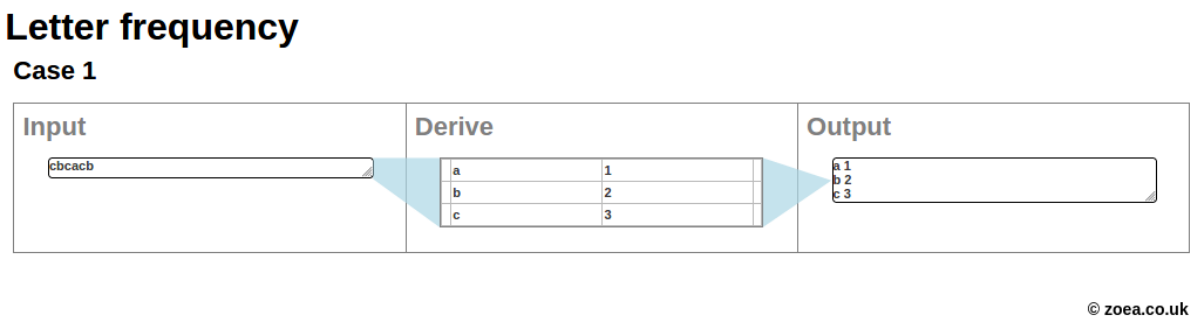
This example would also work if the numbers were represented in other formats such as unary or Morse code, or if alphabetic strings were used.

2.18 Letter Frequency

Task

Count the occurrences of each letter in a string.

Solution



Discussion

This example demonstrates that derive steps can and often do correspond to multiple code operations. To produce the table the input string is split into characters, the number of occurrences of each character is counted and the table is also sorted by the first column. There is no absolute limit to the number of code instructions that can correspond to a single step but a larger number of operations will require more time and resources. It is also harder for anyone looking at the program to figure out what is going on.

This program would also work without the derive step and also without any of the 'c' values.

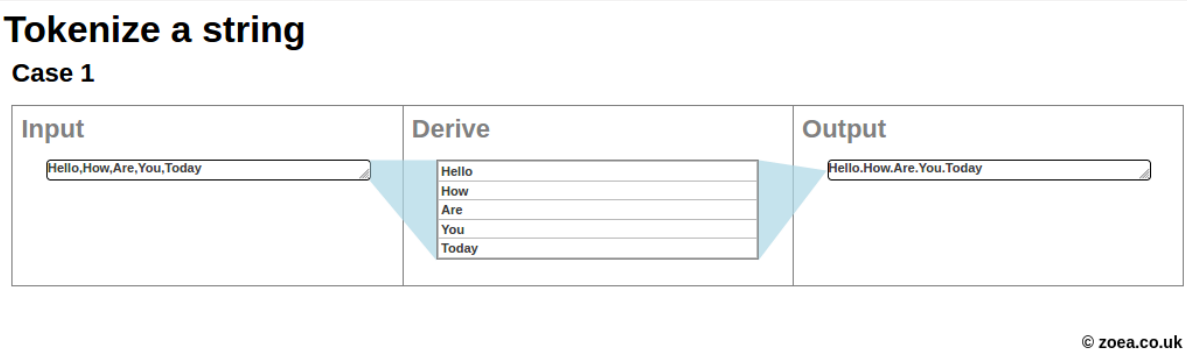
The Rosetta Code description of this task talks about opening a text file. File access in Zoea is currently supported as URLs (see the HTTP example). More sophisticated file and database facilities are on the Zoea development roadmap.

2.19 Tokenise a string

Task

Separate the string "Hello,How,Are,You,Today" by commas into an array (or list) so that each element of it stores a different word. Display the words to the 'user', in the simplest manner possible, separated by a period. To simplify, you may display a trailing period.

Solution



Discussion

Taken together the input and derive resemble the split problem although in this case the delimiter is a comma. Similarly the derive and the output steps constitute a join operation with a period delimiter. This approach of problem decomposition is similar to how the Zoea Visual compiler works at the coarsest level.

Strictly speaking we wouldn't need the derive step if we were happy for Zoea to produce an equivalent program that produced identical results but didn't work internally as described. Without the derive step the result would instead be a program that substitutes comma characters with periods. It so happens that for every distinct program we can define there are a very large number of equivalent programs that produce exactly the same results. We can provide additional derive steps if we are fussy about how the results are produced internally or we can just accept the smallest and simplest of these as a good enough solution.

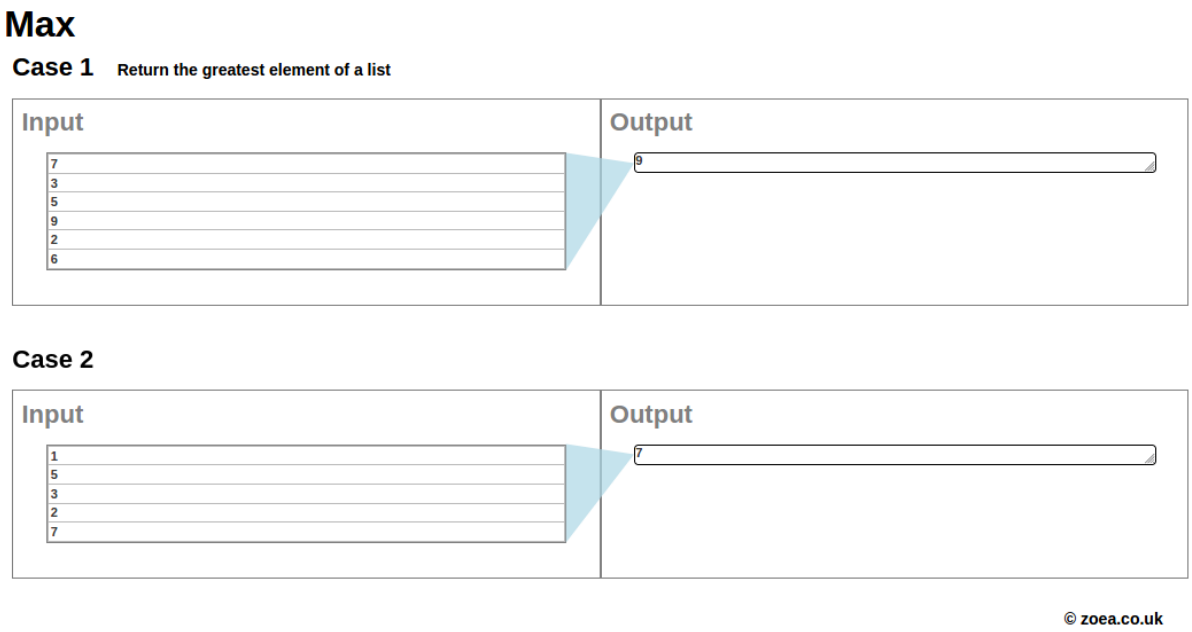
The solution does not include a trailing period however it would be trivial to add one.

2.20 Maximum Element in a List

Task

Return the maximum value from a list of integers.

Solution



Discussion

A single test case where the output is a value returned from a list can correspond to many possible solutions. Without any additional processing these include returning the first, last or nth element. If instead we base element selection on some transformation of the list elements then the possibilities are literally endless.

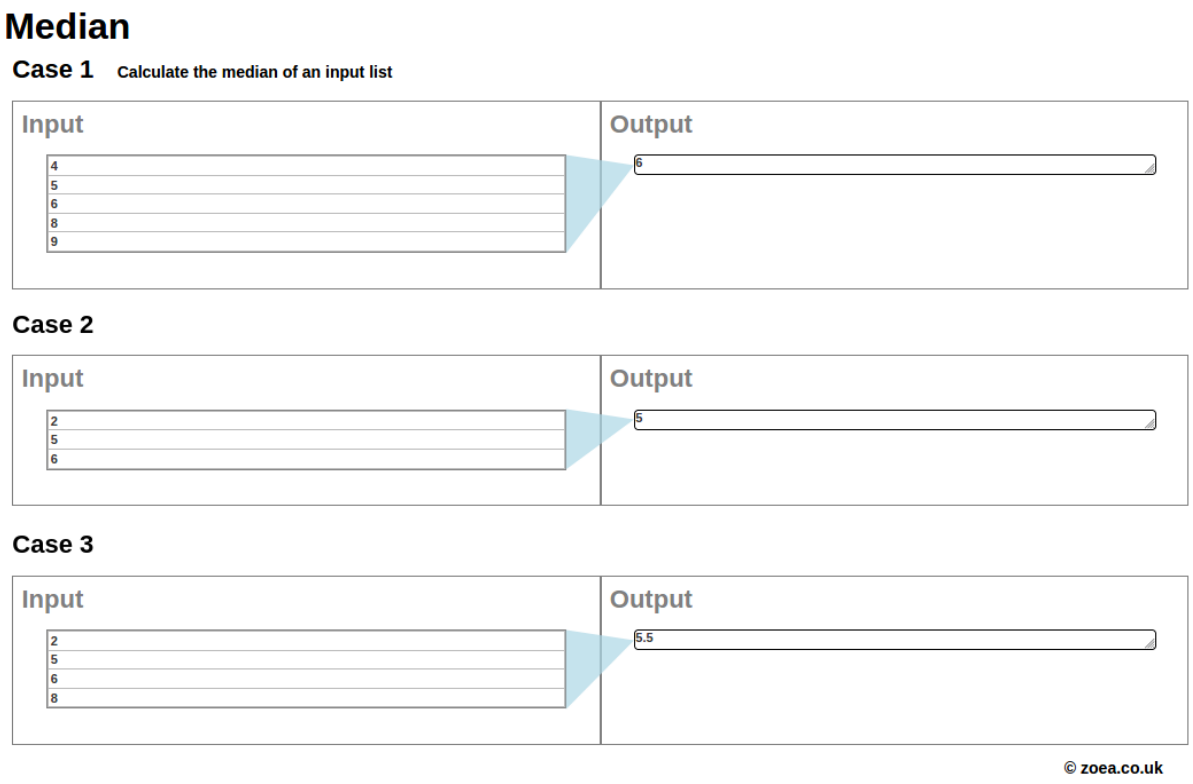
When we select values for the test cases we need to ensure that we are not inadvertently specifying a different program. This is why in this example the returned values are not both the 4^{th} element of the input.

2.21 Median

Task

Calculate the median value of a list of floating point numbers.

Solution



Discussion

The median value is calculated by first sorting a list of numbers and retaining any duplicates. If the length of the list is odd then the median is the middle value from the sorted list. Otherwise if the length of the list is even then the median is the average of the middle two elements in the sorted list.

One obvious thing about the solution is that there isn't a floating point number anywhere in the input. This is deliberate. It is often possible to make the task of producing test data easier without impacting the quality of the solution in any way. In this case the solution produced by Zoea Visual will work for integer inputs and also for floating point inputs or any mixture thereof. Using integers as we have done here allows us to calculate the average value required in the second test case using mental arithmetic. Other problems like average can also be made easier by having for example ten elements in the input.

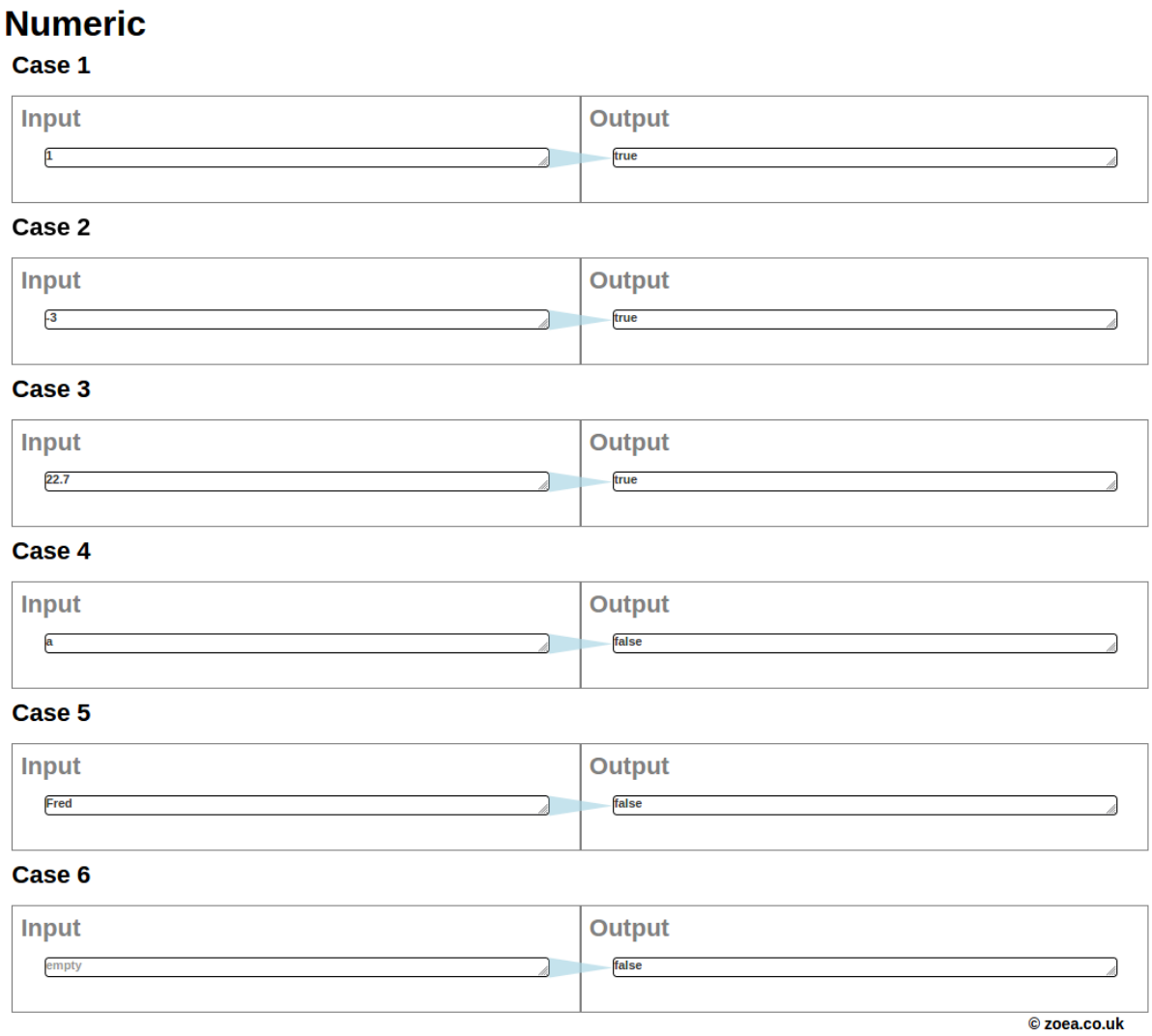
The solution for this problem includes a conditional statement. Zoea is able to identify the processing required for each of the cases but there is no single solution that will work for both of them. In this case the simplest possible condition is correctly identified as input length being odd or even.

2.22 Numeric

Task

Create a boolean function which takes in a string and determines whether it is a numeric string (floating point and negative numbers included).

Solution



Discussion

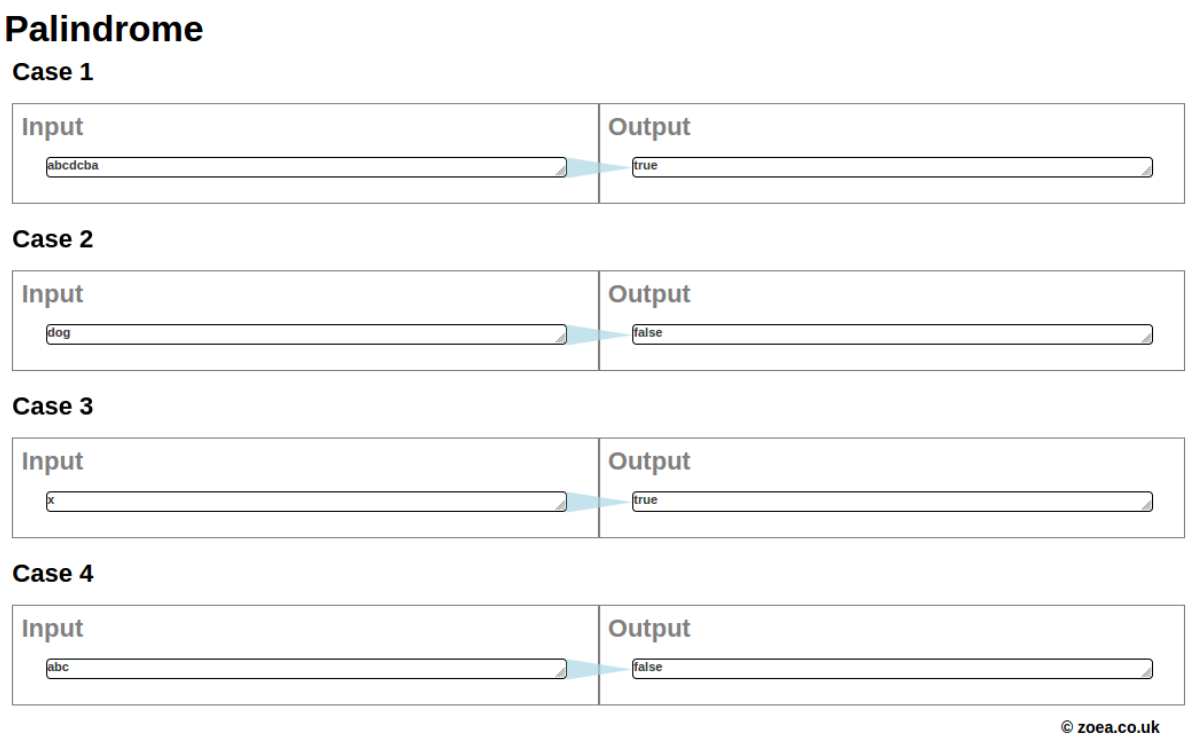
This is a pretty straightforward enumeration of a set of positive and negative examples. It would be possible to have omitted some of the negative cases in order to speed up compilation but this is sometimes a false economy. It is always a good idea to verify the generated solution by tuning it manually with additional test cases. We know that any cases included in the Zoea Visual program will always operate correctly with the generated solution so it's often easier in the long run to specify a program as completely as possible. Producing good test cases is a large part of the skill and effort involved in writing Zoea programs. As a result time spent thinking about what cases and data are required will often produce a better result more quickly.

2.23 Palindrome

Task

Write a function or program that checks whether a given sequence of characters is a palindrome.

Solution



Discussion

A palindrome is a string of one or more characters that is identical if reversed. This problem is a little more complex than the numeric task as it involves a comparison of the input with a mapping of itself.

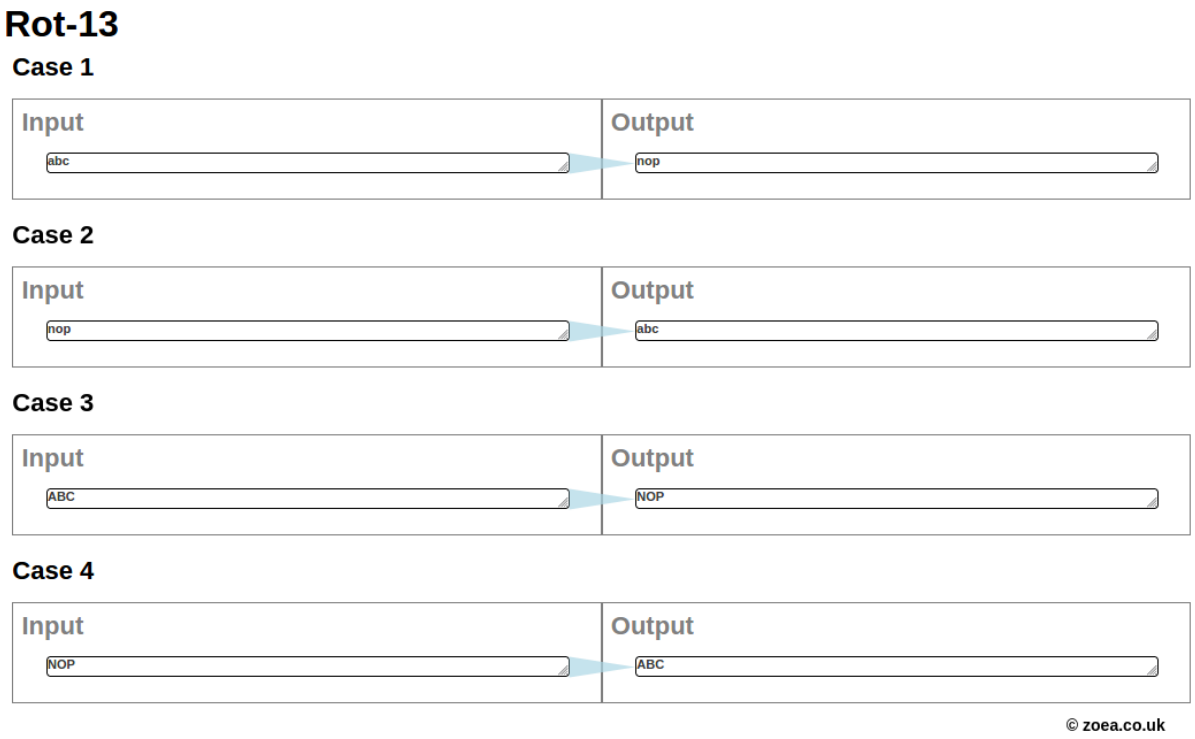
Zoea makes no distinction between functions and programs. Everything in Zoea is a program and every Zoea program could be used as a function if required. At the most basic level test cases can be viewed as a form of boolean function in the sense that they describe the behaviour of a single code path with or without an associated conditional state that selects for that code path. Complete external programs can be integrated as additional instructions through the 'use' tag. In Zoea Visual it is also possible to embed a hierarchy of subsidiary test cases within a single program.

2.24 Rot-13

Task

Implement a rot-13 function. The definition of the rot-13 function is to simply replace every letter of the ASCII alphabet with the letter which is rotated 13 characters around the 26 letter alphabet from its normal cardinal position (wrapping around from 'z' to 'a' as necessary).

Solution



Discussion

This program involves a mapping of the character codes of the input string to produce the output. We need to specify different behaviour for two different ranges for both lowercase and uppercase characters (a..m and n..z). The solution requires conditional logic and the determination of two unspecified quantities: 13 and -13.

Internally some of the knowledge sources that deal with various forms of mapping decompose input and output elements to produce additional synthetic test cases based on the decomposed values. For example, one of the many lines of reasoning explored with regard to case 1 is the set of synthetic cases 'a'→'n', 'b'→'o' and 'c'→'p'. Knowledge sources that handle various kinds of arrays include similar strategies. This is one reason why we often need fewer test cases for mapping problems as composite elements can effectively provide multiple examples of the same logic.

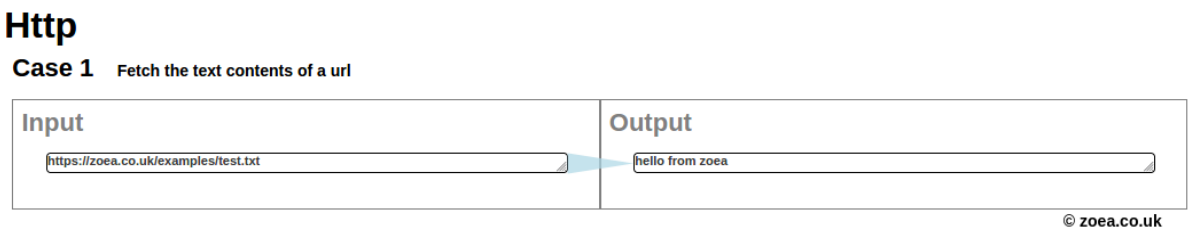
Where we have behaviour associated with ranges of input values Zoea adopts a convention that the lowest value of a range should be specified and the behaviour is assumed to be the same with increasing values until a counterexample is provided. Also the behaviour of the lowest value continues with decreasing values in a similar way. Ranges do not need to be enumerated exhaustively if a subset of examples can demonstrate a recognisable pattern.

2.25 HTTP

Task

Access and print a URL's content (the located resource) to the console.

Solution



Discussion

This task is a nice illustration of one of the fundamental principles of Zoea's design. The software development knowledge that Zoea uses is distributed across a large number of cooperating experts. Each expert proceeds by formulating a set of assumptions about the developer intention based on the test case features that it or one of its peers identifies. Every one of those assumptions may be right but it also may be wrong. Zoea needs to be able to cope with both scenarios.

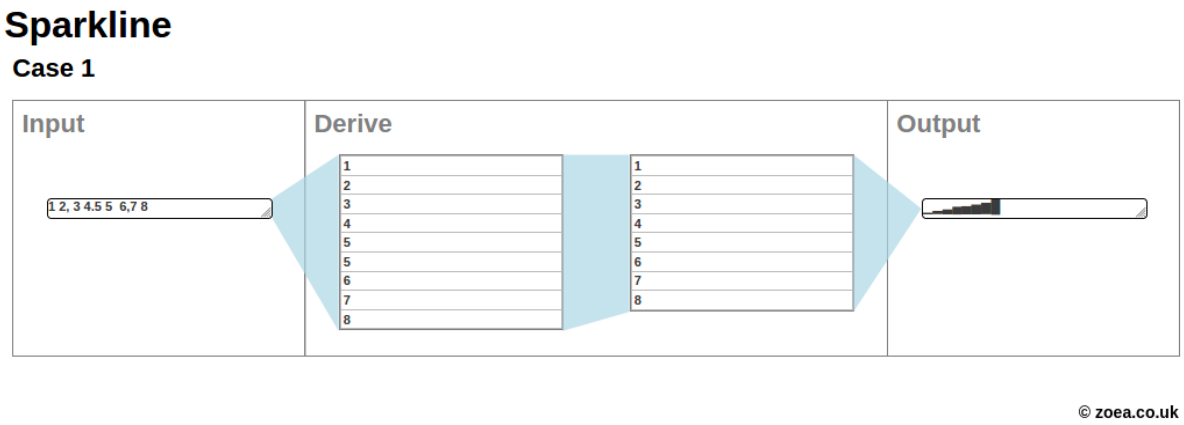
When Zoea detects an object that looks like a URL in some data it is open to multiple possible interpretations. It could for example just be a piece of data such as a string of text. Alternatively, it could be a reference to an internet resource that needs to be fetched as part of the required solution. It is impossible for Zoea to know which if any of these is correct so it tries all possible strategies. To keep things tidy every expert and each hypothesis operates within a different scope. These are provided by a hierarchy of synthetic test cases that are either clones or mutations of their parent test cases. Specific experts are responsible for mapping the results and solutions from descendants at any depth back into their ancestors.

2.26 Sparkline

Task

A sparkline is a graph of successive values laid out horizontally where the height of the line is proportional to the values in succession. Use Unicode values U+2581 through U+2588 to create a program that takes a series of numbers separated by one or more white-space or comma characters and generates a sparkline-type bar graph of the values on a single line of output.

Solution



Discussion

The input field includes all of the examples of single and multiple delimiter combinations. The first derive value is the result of splitting the input string into an array while retaining any consecutive duplicates. The second derive step removes duplicates and the output string is a direct mapping from these values to the corresponding Unicode characters. Note that multiple derive columns are merged in the listing view.

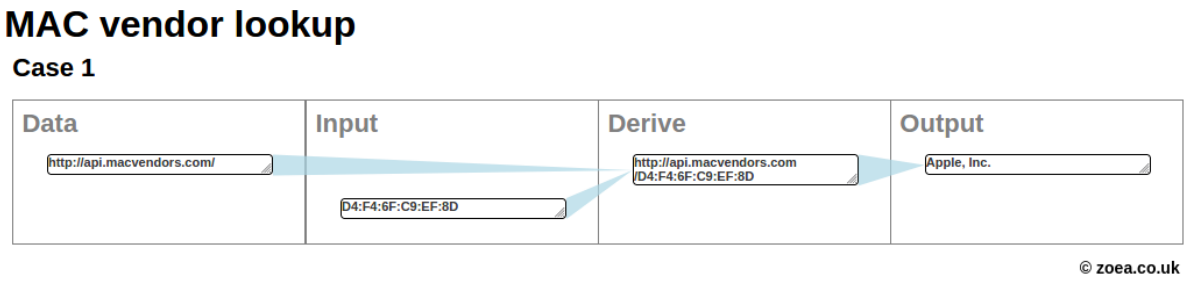
This is still not a complicated problem but it is interesting to compare the size and complexity of the Zoea Visual solution to those in conventional languages. For example the python solution for this task on Rosetta Code is around twenty lines of code and requires a reasonable basis in python or a similar language to understand. With relatively little explanation the Zoea Visual example could be understood by almost anyone.

2.27 MAC Vendor Lookup

Task

Interface with an API that exists on the internet and retrieve the device manufacturer based on a supplied MAC address.

Solution



Discussion

In this solution we have defined the URL for the service as data so it can be changed in the future without having to recompile the program. We could also have used a test URL and instead defined the real URL as runtime data. See the Zoea Visual manual [2] for details on how to do this.

In this case the input MAC address is simply concatenated to the service URL to produce a request. Note that the derive value is really a single line of text but it has been wrapped by the listing view. The output value in this example is the whole of the HTTP response document. If instead an HTTP header value or just part of the document was required then this could be described using additional derive steps.

Acknowledgements

Most of the examples in this document either come from or are based on tasks from the Rosetta Code web site [1]. Zoea is a trademark of Zoea Ltd. All other trademarks are the property of their respective owners. This document is copyright Zoea Ltd 2020. All rights reserved. Visit https://zoea.co.uk/ for more information.

References

[1] Rosetta Code. Available at: https://rosettacode.org/

[2] Zoea Visual Manual. Available at: https://zoea.co.uk/language/documents.html