CHAPTER 10

Working with Text

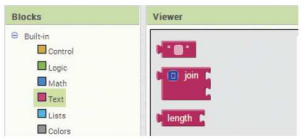
Topics

- Concatenating strings
- Comparing strings
- Trimming a string
- Converting case
- Finding a substring
- Replacing a substring
- Extracting a substring
- Splitting a string

Concatenating Strings

- To concatenate strings means to join strings together.
- To concatenate two strings, use the `Text` **`join`** block.
- The **`join`** block has parameter sockets that will allow you to join multiple text blocks together.

Figure 10-1 The Text `join` Block (Source: MIT App Inventor 2)



Concatenating Strings

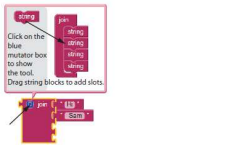
- The `join` block appends text in order.
- Consider Figure 10-2. The first text block is “Hi” and the second is “ Sam”(noticed the preceding space). The results from this `join` will be “Hi Sam”.

Figure 10-2 Concatenation (Source: MIT App Inventor 2)

Concatenating Strings

- The `join` block makes use of the *mutator* tool and allows for more slots of text. See figure 10-3.

Figure 10-3 Using the Mutator to Add More Slots (Source: MIT App Inventor 2)



Concatenating Strings

- Any block that is plugged into the `join` block will be treated as text.
- If both arguments are numbers, they are still treated as strings.
- Concatenating the string 12 and the string 17 will result in the string 1-2-1-7, not the number 29.

Concatenating Strings

Concatenating String Literals

A literal string is a string made up of a sequence of characters. See Figure 10-4.

Figure 10-4 Join Two String Literals (Source: MIT App Inventor 2)

Notice that there is no space between *Hello* and *World.*

Concatenating Strings

Concatenating String Literals

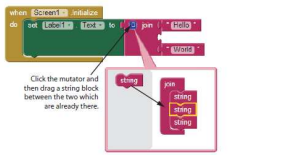
- We can add a space between *Hello* and *World* a few different ways.
- One way is to add a space at the end of the word *Hello*. That may not be the best solution.
- An additional solution would be to create another string literal containing just a space and then join it in the middle of the other two.

Concatenating Strings

Concatenating String Literals

To join three string literals with App Inventor, use the mutator to build a join block with three slots.

Figure 10-5 Joining Three String Literals (Source: MIT App Inventor 2)

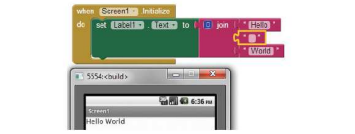


Concatenating Strings

Concatenating String Literals

Once you have the slots that you need, add a single space text block. See Figure 10-6

Figure 10-6 Adding a Space (Source: MIT App Inventor 2)



Concatenating Strings

Concatenating String Literals

To complete the join blocks shown in Figure 10-6 follow the steps:

1. Make a text block with a single space.
2. Join the word *Hello* with the space.
3. Use the *mutator* tool to add a third slot to the join block.

Concatenating Strings

Concatenating Variable Strings

If you have variables that contain strings you plug the variable value block into the join block.

Figure 10-7 Joining Variable Strings (Source: MIT App Inventor 2)

Concatenating Strings

Concatenating Variable Strings

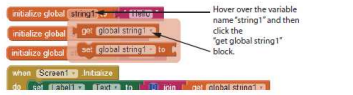
- We join the variable `String1`, `String2` first, and then join those with `String3`.
- These blocks are variables and not literals, therefore the value of each variable is used to create the resulting string *Hello World*.
- To find the value of a variable, you hover over the name of the variable in the declaration block.

Concatenating Strings

Concatenating Variable Strings

In Figure 10-8 explains how to find the get global `String1` variable block.

Figure 10-8 Finding a Variable's `get` Block (*Source:* MIT App Inventor 2)



Concatenating Strings

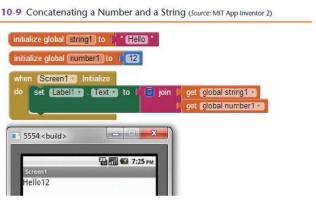
Concatenating Strings with Numbers

- As previously stated, when you use the `join` block for concatenation, all data plugged into it will be treated as text.
- To demonstrate the effort of concatenating strings with numbers, see Figure 10-9.

Concatenating Strings

Concatenating Strings with Numbers

Figure 10-9 Concatenating a Number and a String (*Source:* MIT App Inventor 2)



Even though the variable `Number1` is a number, the concatenation still works, but the number is treated as text.

Concatenating Strings

Concatenating Two Numbers

- When you add the numbers 12 and 17 the result is 29.
- When you concatenate the string 12 with sthe string 17, the result will be the string 1-2-1-7. This is not the same as the number 1217.
- Figure 10-10 demonstrates concatenating numbers.

Concatenating Strings

Figure 10-10 Concatenating Two Numbers (*Source:* MIT App Inventor 2)

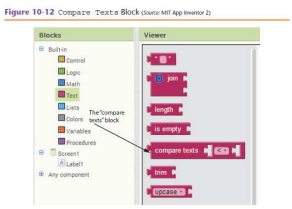
Comparing Strings

- We can compare two strings to determine whether they are equal, whether one is greater than the other or whether one is less than the other.
- An example of comparing two strings for equality would be password verification.
- An example of using the greater than or less than comparison would be to alphabetize a list of names.

Comparing Strings

`Compare Texts` Block

The compare text block allows us to determine whether two strings are equal or whether one or the other is less than or greater than as seen in Figure 10-12.



Comparing Strings

`Compare Texts` Block

To change the operator so that it tests for equality, less than, or greater than, you click on the down arrow in the middle of the block.

Figure 10-13 Change the Operator (Source: MIT App Inventor 2)

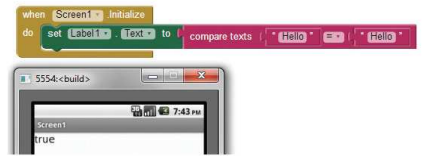
Comparing Strings

Equal Strings

- Just like the Text `join` block, if you plug in a number or other data type, the `compare texts` block will treat it as text.
- For two strings to be equal they must be identical. This includes case sensitivity.
- Figure 10-16 shows an example of using the `compare texts` block to evaluate two strings that are identical and returns the value of true to the `Label1.Text` property.

Comparing Strings

Figure 10-16 Equal Strings (Source: MIT App Inventor 2)



Comparing Strings

Greater Than or Less Than Comparisons

- In computing, every printable character has an associated number represented in the ASCII (American Standard Code for Information Interchange) table.
- ASCII is a set of 128 numeric codes that represent the English letters, punctuation marks and other characters.

Comparing Strings

Greater Than or Less Than Comparisons

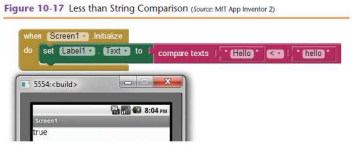
- The ASCII code for the uppercase letter “A” is 65.
- The lowercase “a” is 97.
- Therefore the lowercase “a” is greater than the uppercase “A”.

Comparing Strings

Greater Than or Less Than Comparisons

- When we compare *Hello* and *hello*, “H” and the “h” are compared first.
- Figure 10-17 shows that *Hello* Computes to a value less than *hello*.

Figure 10-17 Less than String Comparison (Source: MIT App Inventor 2)



Trimming a String

- Trimming a string means to remove any leading or trailing spaces from it.
- Unwanted spaces can be a result of human error (typos).
- This block will remove both leading and trailing spaces from a string and return the resulting string.

Trimming a String

Figure 10-29 Using the `trim` Function Block (Source: MIT App Inventor 2)

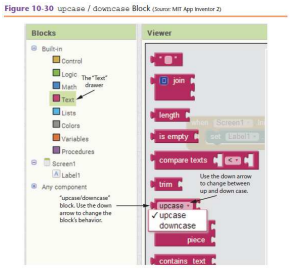
Notice in this figure that there are a few leading and trailing spaces surrounding the literal string.

Trimming a String

- It is also important to note that trimming only removes leading and trailing spaces and not any spaces inside the string.
- Trimming the *Hello World* text block in Figure 10-13 does not remove the space between *Hello* and *Hello World*.

Converting Case

- Converting case means to convert a lowercase letter to an uppercase letter or vice versa.
- The upcase / downcase block is the found in the *Text* drawer.



Converting Case

If you convert a string to uppercase with the upcase block, the function will return the string with all capital letters. *Hello World* will convert to *HELLO WORLD*.

Finding a Substring

- A substring of a string is a set of characters that exists as part of that string.
- App Inventor provides two blocks that can help us find substrings.
  1. One is used to determine whether the string contains a substring (the `contains` block).
  2. The other will tell us where the substring begins (the `starts` at block).

Finding a Substring

The contains function block:

- Returns a Boolean value based on whether or not the substring exist in a string.
- Requires two parameters, `text` and `piece`.

Finding a Substring

Figure 10-37 Contains Example (Source: MIT App Inventor 2)

This function would return false because the string *Grapes* does not exist in the string *Apples and Oranges*.

Finding a Substring

- The next block that is useful for finding substrings is the `starts at` a function block.
- It requires the same two parameters as the `contains` block, `text` and `piece`.

Figure 10-38 `Starts at` Block (Source: MIT App Inventor 2)

- The function will return a number representing the position in the `string` (`text` parameter) where the substring (`piece` parameter) starts.

Finding a Substring

Figure 10-39 Substring Example (Source: MIT App Inventor 2)

- The function call in this figure will return the number 8 because the substring *Orange* exists and it starts in the eigth position.
- Both the `starts at` and `contains` blocks are *case sensitive*.

Replacing a Substring

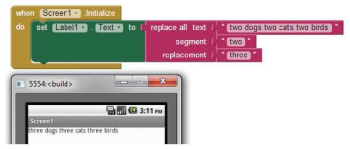
- App Inventor has a `replace all` block that returns a copy of a string.
- The function call in Figure10-55 will return *barking up the right tree*.

Figure 10-55 Replace All Block (Source: MIT App Inventor 2)

Replacing a Substring

- If there were more occurrences of the word *wrong*, all of them would be replaced with the word *right* as in Figure 10-56.
- The `replace all` block is case sensitive.

Figure 10-56 Replace All Example (Source: MIT App Inventor 2)



Extracting a Substring

- You can also extract a substring from a string.
- To do this you will need to know what the starting point and the length of the substring.
- The segment block will allow you to extract a substring by giving it three parameters.

  `text` – the entire string.

  `start` – the starting position of the substring.

  `length` – the length of the substring.

Figure 10-57 Segment Block (Source: MIT App Inventor 2)

Extracting a Substring

- Let's say you have a string of products that contains the name, product number, and price.
- The product number begins with the letters PN and is followed by 6 digits.
- To extract the product number from the string we first find where the substring begins by using these `starts at` block.

Extracting a Substring

Figure 10-58 Extracting a Substring (Source: MIT App Inventor 2)

Extracting a Substring

We first created a variable `ProductString`.

In the `Screen.Initialize` event we populate a label with the product number.

To extract the product number we used the segment block.

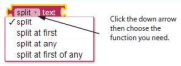
To determine the starting point, we use the `starts at` block.

Splitting a Substring

App Inventor allows us to split strings into list items by providing us several split functions.

Figure 10-59 The `split` Block (Source: MIT App Inventor 2)

Figure 10-60 Accessing the Other Functions (Source: MIT App Inventor 2)



Splitting a Substring

- Two of these functions, the `split at first` and the `split at first of any`, return a simple two-item list.
- `split`, `split at any`, and `split at spaces`, return a list that can be more than two items.
- The *division-point* is the parameter value you supply where you want to split the string.

Splitting a Substring

- For the `split at first` and `split` blocks, the division point you supply will be a single string or character, like a comma, period, or word.
- If you use the `split at first` of anti-or the `split at any` blocks, your division-point parameter will be a list.

Splitting a Substring

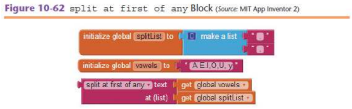
`split at first`

- The split functions that contain the words “at First” will return the two-item list.
- The first element will be A and the second element will be E, I, O, U.

Figure 10-61 `split at first` Block (Source: MIT App Inventor 2)

Splitting a Substring

Figure 10-62 `split at first of any` Block (Source: MIT App Inventor 2)



- This block will return a two-item list.
- The first step is to make a list of division-points and store it to a variable.

Splitting a Substring

- Use that list variable as the division-point in the at parameter of the block.
- The first occurrence of a comma or period happens to be between the E and I.
- The first element will be A E and the second element will be I, O, U, y.
- You cannot use a space as a division point.

Splitting a Substring

`split`

- Next, let's look at the `split` and `split at any` functions.
- The difference between these and the `split at first` blocks is that these functions will split the string at all locations of the division-point.
- The `split` function block shown in Figure 10-63 will return a list of five elements (A E I O U).

Figure 10-63 `split` Block (Source: MIT App Inventor 2)

Splitting a Substring

Figure 10-64 `split at any` Block (Source: MIT App Inventor 2)

- Notice that both commas and periods separate the vowels.
- The returned list from the `split as any` block will also be the five elements (A E I O U).

Splitting a Substring

`Split at Spaces`

App Inventor also provides a split at spaces function that will split a text block by spaces .

Figure 10-65 Split at Spaces (Source: MIT App Inventor 2)

In this example, the `split at spaces` function block will return a list of five elements (A E I O U) with no spaces.

Splitting a Substring

Length of a String

- App Inventor provides two blocks that will help us determine the length of a string.
- The `is empty` block still let us know if the string is empty.
- The `length` text block will return a number.

Figure 10-66 `is text empty` Block (Source: MIT App Inventor 2)

- In this example the function would return false because the name is not empty.

Splitting a Substring

Length of a String

Figure 10-67 `length` Block
(Source: MIT App Inventor 2)

This function will return to the number 5 because there are 5 characters in the name *Sally*.