```
// Word Parser Demo (5 Oct 2021)
//
// ******************************
//
// Note that everything is
// inside the CREATE event in my
// gamemaker studio version of this
// code, because I only want it to run once.

// Questions?  Let me know.
// gwl

// ******************************

// Let's start with the sentence we want to pull apart.
// punctuation can cause trouble:  I'm deleting it by hand
// from this example.  We can discuss other approaches to
// fixing punctuation in person.
mySentence = "It is a far better thing I do now than I have ever done";

// Sometimes, cheating is the best approach.  In this code,
// I look for spaces (" ") as a way to divide up words.
// But since there's no space at the end of the sentence,
// my code doesn't grab the last word!  That's easy to fix,
// but for now, we can cheat by sticking an extra space
// at the end of the sentence.  In development parlance,
// this is called a "hack."
mySentence = mySentence + " ";

// Now:  How many letters are in my sentence?
// Since computers can't be relied upon to identify individual
// words (in English no less) on their own, we're in for a
// long slog, as we need to move LETTER BY LETTER through
// the sentence.  Luckily, the computer is fast.
// So in order to ITERATE through that sentence, letter by
// letter, and look for spaces, I need to know how many
// letters are actually IN that sentence.  (Again, the computer
// "understands" the idea of letters/characters; the idea
// of words, though?  Not so much.)

letterQty = string_length(mySentence);

// Planning ahead:  I'm going to need a place to store
// each of the words I find.  So we'll use an array --
```

```
// which we've relied upon before.  An array is like a variable
// on steroids.  This statement creates an empty array
// called "myWordCollection".

myWordCollection = [];

// Last thing before we get into the FOR() loop:
// I want a piece of scrap paper upon which to have the
// computer keep a running tally of each word as we're
// assembling it.

myTempWord = "";

// Now we ITERATE, looping through each and every
// character inside the sentence.  The FOR() loop just
// keeps track of where we are in the sentence, and
// lets us perform an operation on each letter.

// AND AGAIN, that's the point:  Computers are dumb.  We
// can't ask it to do complicated things.  We've got to
// be very simple in our requests.  In this case, we just
// keep asking, "OK... do you see the next letter in the
// sentence?  Is that letter a space?" etc.  It is like
// talking to a very small child:  You can not take much
// for granted.

// the FOR LOOP:  We start with letter number 1
// and iterate through the total number of letters.
// We'll use the variable "i" (as in "index") to
// keep track of that value.
for (var i=1; i <= letterQty; i++){

    // NOW INSIDE THE FOR LOOP.  The variable
    // "i" is somewhere between 1 and letterQty.
    // Let's get to work.

    // First up:  let's ask the computer to
    // look at a specific letter (position i)
    // in our sentence.  We'll call that letter
    // myLetter.  In this case, when "i" is
    // 1, myLetter will be "I".  When it is 2,
    // myLetter will be "t".  (See mySentence, line 6).

    myLetter = string_char_at(mySentence,i);
```

```
// We've got the letter.  The computer has no idea
// what it means.  So we've got to do the work here:

// FIRST:  IS THE LETTER A "SPACE"?  Because if it is,
// that means we've come to the end of a word!

if (myLetter == " "){

    //Ah-ha.  It was a space!  That means we need
    // to save the word we've built and then move
    // on to the next word (i.e., wait for the next
    // space).  To save the word to our array
    // I use this fancy function, array_push().  It
    // just adds a new word to my growing collection.
    // I use it by supplying two arguments:  First,
    // Q: What is the array I want to use to store the
    // word? (A:  myWordCollection); second, what do I
    // want to store therein?  (A: myTempWord)

    array_push(myWordCollection, myTempWord);

    // That's it.  It is now stored safely.  We
    // just need to destroy the scrap paper so we
    // can start building a new word (because, again,
    // we encountered a space, which -- in English --
    // we use to seperate words.)

    // recycle the scrap paper:
    myTempWord = "";

// ELSE -- So we finished the first part of the
// IF statement, where we told the computer what to do
// when we found a space.  But what if we don't find
// a space?  That means we're in the middle of a word!
// which means we need to jot that letter down on our
// scrap paper.
} else {

    // AGAIN:  This line is only allowed to run when
    // we find a letter and not a space.
    // We add that letter to the temporary
    // word and then we're done.  Close off the curly
    // braces, and let the computer keep iterating through
    // the rest of the letters.
    myTempWord = myTempWord + myLetter;
```

```
    // this curly brace closes the "else" part of the
    // if statement that began on line 83.  Remember:
    // if there are 100 letters, the computer will ask the
    // question "Is this letter a space?" 100 times.
    }

// This curly braces closes off the FOR() loop.  When it
// hits this symbol, it goes back to the FOR() loop start
// (line 62) and adds 1 to "i".  Then it asks "is i STILL less
// than or equal to the total number of letters in my sentence?"
// If it is, then it repeats the loop (and all the stuff inside
// it) AGAIN.  and AGAIN.  and AGAIN.  Until finally i is GREATER
// than the total number of letters in my sentence, which ends
// the ordeal, and dumps us unceremoniously into the line
// after this curly brace.
}

// THAT IS IT.  NOW WE HAVE AN ARRAY
// jammed full of words.  Arrays are so amazingly useful,
// but they are really nothing more than tiny
// spreadsheets -- just simple tables.  Here's what ours
// "looks" like.

// NOW WE NEED TO FIGURE OUT WHAT WE WANT TO DO WITH
// OUR NEWLY PARSED DATA.  That's not the purpose of this
// exercise, though:  We just wanted to parse it out.

// ********************

// Example

// ********************
// If each word were written on a card in a deck of cards,
// we could now draw a random word card:

// first, get the size of the array (how many words/cards)?
var cardTotal = array_length(myWordCollection);
// But it's a trap!  Kinda!  The function array_length()
// returns the number of entries in our array (14), but since
// the first entry is actually entry number 0...
// that means that myWordCollection[14] doesn't exist!
// The last word/card is myWordCollection[13] ("done").
// If we ask for number 14?  Errors everywhere.  So
// we have to stay vigilant.  Keep watching the skies.
```

0	It
1	is
2	a
3	far
4	better
5	thing
6	I
7	do
8	now
9	than
10	I
11	have
12	ever
13	done

```
// With that in mind, pick a random card
// ensure really random number:
randomize();

// grab a card!
var cardID = random_range(0,cardTotal-1);
// Show me the card!
show_message("Random word card: \n" + myWordCollection[cardID]);

// ********************
// Example
// ********************

// Let's write the sentence backwards!
// We'll need a scrap of paper to use to assemble the
// sentence as we go.

var sentence = "";

// And then we'll just count backwards, from
// the last word to the first (from 13 to 0!), adding
// each word to our sentence, PLUS A SPACE
// TO SEPARATE IT from the other words.

for (var i = cardTotal-1; i>=0; i=i-1) {

    // Grab the word from our array:
    var temp = myWordCollection[i];

    // Add it to the sentence!
    sentence = sentence + temp;

    // Add a space after each word!
    sentence = sentence + " ";

// This curly brace indicates the end of the FOR()
// loop.  It sends us back to line 188, where we
// subtract 1 from i and then ask "IS i STILL EQUAL TO
// OR GREATER THAN 0?"  If yes, then we do the code
// block again, and again, and again, until "i" is
// finally LESS THAN 0 (it'll be -1), which causes us
// to end the loop.
}
```

```
// NOW we show what we created with all that work!
show_message("Backwards: \n"+sentence);


// **************
// EXAMPLE
// **************

// Let's do Data Science!
// What's the average length of words?

// our running total for the average
var workingTotal = 0;

// Loop through each word and get the size
for (var i=0; i<array_length(myWordCollection); i++) {
    var tempWord = myWordCollection[i];
    var size = string_length(tempWord);
    workingTotal = workingTotal + size;
}

// Now divide total by number of words!
// This is MORE EXCITING when there are more
// sentences, words, variety, etc.
// (But even then it isn't really that exciting).

var outcome = workingTotal / array_length(myWordCollection);

// Note that since "outcome" is a number,
// I need to use the "string()" function on it
// so that it can appear in my message.
show_message("Average word length: \n" + string(outcome));

// ...Note that the answer seems
// suspiciously even ("3"?  Really?)
// but that is really the average.


// 2021 Garrison Winfield LeMasters
// Miami OH / ETBD / Games + Simulations
```