10 ways to reverse a string in JavaScript

1: Decrementing for-loop with concatenation

```
function reverse(s) {
  var o = '';
  for (var i = s.length - 1; i >= 0; i--)
    o += s[i];
  return o;
}
```

The original way that I achieved the intended result was to use a decrementing for-loop that appended each character of the input to a new string in reverse order. I was able to access the parsed strings individual characters similar to the way you would reference an array's items.

2: Incrementing/decrementing for-loop with two arrays

```
function reverse(s) {
  var o = [];
  for (var i = s.length - 1, j = 0; i >= 0; i--, j++)
    o[j] = s[i];
  return o.join('');
}
```

Another way I formed to reverse a string was to create an empty array and iterate over the length of the string with both incrementing/decrementing counters. The array position uses the incrementing counter where as the parsed in string uses the decrementing one. Finally the created array is joined into a single string and returned.

3: Incrementing for-loop with array pushing and charAt

```
function reverse(s) {
  var o = [];
  for (var i = 0, len = s.length; i <= len; i++)
    o.push(s.charAt(len - i));
  return o.join('');
}
```

The above example is a modification of the second example. Instead of using two counters however we use one incrementing value that gets deducted from the total length of the parsed in string. This calculated value determines the position of the next character to be pushed onto the new array (using the ‘push()’ function instead of ‘[]’). The other difference from the last example is that it uses the strings ‘charAt()’ method instead of its array capabilities.

4: In-built functions

```
function reverse(s) {
  return s.split('').reverse().join('');
}
```

This implementation takes advantage of the ‘reverse()’ method provided by the Array prototype. First it splits the string into a real array, then calls the ‘reverse()’ method and finally returns the joined array.

5: Decrementing while-loop with concatenation and substring

```
function reverse(s) {
  var i = s.length,
      o = '';
  while (i > 0) {
    o += s.substring(i - 1, i);
    i--;
  }
  return o;
}
```

Using a decrementing while-loop I was able to implement this method. Again harnessing concatenation, I was able to achieve the iteration through the string in a similar fashion to the for-loop used in the first two examples. I was then able to use the strings ‘substring()’ function to retrieve each desired character.

6: Single for-loop declaration with concatenation

```
function reverse(s) {
  for (var i = s.length - 1, o = ''; i >= 0; o += s[i--]) { }
  return o;
}
```

This is most likely my favorite implementation, due to its unnecessary cryptic nature. Using only a single for-loops parameters, I was able to decrement through the parsed in string and concatenate each character to a new string to return.

7: Recursion with substring and charAt

```
function reverse(s) {
  return (s === '') ? '' : reverse(s.substr(1)) + s.charAt(0);
}
```

The above example recursively calls itself, passing in the inputted string, excluding the first character on each iteration, which is instead appended to the result. Iterating through this process until no input is present (the base case) results in a reversed string.

8: Internal function recursion

```
function reverse(s) {
  function rev(s, len, o) {
    return (len === 0) ? o : rev(s, --len, (o += s[len]));
  };
  return rev(s, s.length, '');
}
```

This is another example of using recursion to reverse a string. The implementation above uses an internal function, which is first called by the outer function, parsing in the inputted string, its length and an empty string. The internal function is then recursively called by itself until the string length has been decremented to zero - at which time the originally empty parsed in string has been concatenated with the inputted string characters in reverse.

9: Half-index switch for-loop

```
function reverse(s) {
  s = s.split('');
  var len = s.length,
      halfIndex = Math.floor(len / 2) - 1,
      tmp;
  for (var i = 0; i <= halfIndex; i++) {
    tmp = s[len - i - 1];
    s[len - i - 1] = s[i];
    s[i] = tmp;
  }
  return s.join('');
}
```

I found this method to be a very effective way of reversing a string, highlighting its benefits when processing large strings. The strings half-point is first calculated and then iterated over. Upon each iteration the upper half's value (calculated by deducting the current position by the string length) is temporary stored and replaced by the lower half's value. The temporary value then replaces the lower half's value to finally result in a reversed string.

10: Half-index recursion

```
function reverse(s) {
  if (s.length < 2)
    return s;
  var halfIndex = Math.ceil(s.length / 2);
  return reverse(s.substr(halfIndex)) +
         reverse(s.substr(0, halfIndex));
}
```

The final method I wish to show uses the same ideology as the last implementation (half-indexing) but instead relies on recursion to reverse the string instead of a for-loop.

Follow me for such info

https://www.linkedin.com/in/priya-bagde/

https://github.com/priya42bagde

https://www.youtube.com/channel/UCK1_Op30_pZ1zBs9l3HNyBw/videos