Assignment 2: Colorized and Formatted JSON¶

JavaScript Object Notation, commonly called JSON, is a popular data interchange format. Many programmers like reading JSON that is consistently formatted and colored JSON, and so for this assignment, your task is to write a Go program that takes any *valid* JSON as input (from standard input) and outputs (to standard output) HTML that transform the JSON as follows:

- all tokens are consistently colored in a pleasing way
- the formatting is neat and consistent

Importantly, you can assume that the JSON given as input to your program is **always valid** without any errors. So your program doesn't need to check for invalid JSON.

Please refer to the JSON grammar on the JSON home page for exact syntax details.

Colorization¶

The following groups of JSON tokens should each have a unique color:

- { and }
- [and]
- :
- ,
- true, false, and null
- strings; escape characters within a string, like \n or \u8a3e should be given a different color
- numbers; its fine if all the digits, and the (optional) . and e/E/+/- characters are the same color (although you can give them different colors if you think it looks better)

So your output will need at least 7 different colors. The exact colors are up to you — try to make it look good!

Formatting¶

While the input JSON given to your program is always valid, it might not be neatly and consistently formatted. So, in addition to coloring the tokens, your program should put consistent whitespace around them. For example, the JSON fragment `{"s":[2, 3], "a < b && a >= c":true}` could be nicely formatted like this:

```
{
  "s" : [2, 3],
  "a < b && a >= c" : true
}
```

The formatting rules can be relatively simple. For instance, you might use rules like this:

- { and } tokens always go on their own line
- there is a space before and after each :
- pairs in curly-braces go on their own lines, and if their values happen to also have curly-brace expressions, then they should be indented further in

The exact formatting rules are up to you, but try to make the JSON as easy to read as possible.

Input and Output¶

Your program should read its input from a file named `a2.go` that is passed as an input to your program when it runs, e.g.:

```
$ go run a2.go input.json
```

Use `os.Args` to get the name of the input file. If no file is provided, then it's fine if your program ends with an error message.

The HTML output should be printed to standard output using `fmt.Printf` statements. In Linux/Unix, you can easily re-direct standard output to a file using the `>` operator:

```
$ go run a2.go input.json > json.html
```

Now you can view the file `json.html` in a web browser.

Using HTML¶

To preserve indentation, please wrap your output in `span` tags like this:

```
<span style="font-family:monospace; white-space:pre">

... source code goes here ...

</span>
```

To color a token, use a `span` tag with the `color` style set. You can select colors in a couple of different ways, e.g.:

- `<span style="color:red">{"</span>`
- `<span style="color:rgb(255, 0, 0)">}"</span>`

A few characters must be replaced by special symbols to be displayed properly in HTML. In particular:

- Replace `<` with `&lt;`
- Replace `>` with `&gt;`
- Replace `&` with `&amp;`
- Replace `"` with `&quot;`
- Replace `'` with `&apos;`

So, for example, the JSON fragment `{"s":[2, 3], "a < b && a >= c":true}` should be written like this in HTML:

```
{&quot;s&quot;:[2, 3], &quot;a &lt; b &amp;&amp; a &gt;= c&quot;:true}
```

With coloring tags, it can get pretty messy. For example, the JSON fragment `"s":[2, 3]` could look like this:

```
<span style="color:green">&quot;s&quot;</span>
<span style="color:orange">:</span>
<span style="color:red">[</span>
```

```
    <span style="color:yellow">2</span>
    <span style="color:blue">,</span>
    <span style="color:yellow">3</span>
<span style="color:red">]</span>
```

The HTML generated by your program **doesn’t** need to be human-readable. What matters is that web browsers can display it properly.

Hints¶

- You should divide your program into two main parts: a JSON scanner that returns a list of tokens, and an a colorizer/formatter that transforms a list of tokens into neatly HTML that is nicely formatted and colored.
- Be careful with characters in strings, e.g. in the JSON fragment `{[1, 2, 3], "{key:val}"}`, the `{`, `}`, and `:` characters inside the string should be colored like any other character in a string. Also, if `\"` occurs in a string, it is an escape character, and the quote-mark does not end the string.
- JSON numbers have a few different formats you need to handle. For example 45, -23, 102.332, -52.01e-355, and -475246256524654 are all valid JSON numbers.
- While writing scanners is not extremely difficult, they do take time and they can be tricky due to “off by 1” errors that arise when checking for the ends of tokens and strings. A good approach is to test your programs with small examples as you go, making sure that each part of your program works before moving on to the next.
- Don’t worry too much about the size of the resulting output. Stick to the basic style of HTML/CSS in the examples so that it displays properly in any browser.