COMP[29]041 18s2

Week 11 Weekly Test

COMP[29]041 18s2

Sample Answers

Test Conditions

These questions must be completed under self-administered exam-like conditions.
You must time the test yourself and ensure you comply with the conditions below.

- You may complete this test in CSE labs or elsewhere using your own machine
- You may complete this test at any time before **Saturday 5 January 19:00**
- The maximum time allowed for this test is 1 hour + 5 minutes reading time.
- You may first use 5 minutes to read the questions (no typing)
- You then must complete the test within 1 hour and submit your answers with give.
- You must complete the questions alone - you can not get help in any way from any person.
- You can not access your previous answers to lab or tut questions.
- You can not access web pages or use the internet in any way.
- You can not access books, notes or other written or online materials.
- You can not access your own files, programs, code ...
- You can not access COMP2041 course materials except for language documentation linked below.

You may access this **language documentation** while attemting this test:

- Shell/Regex/Perl quick reference
- Javascript quick reference
- full Perl documentation
- Python quick reference
- C quick reference
- full Python 3.6 documentation

You may also access manual entries (the man command) Any violation of the test conditions will results in a mark of zero for the entire weekly test component.

Return the sum of a list

Your task is to write a Javascript function **sum** which takes as its first argument a list of strings.
Each string will contain only digits ([0-9]).

Your function **sum** should return the sum of the integers.

For example given this list:

```
["17", "10", "17"]
```

Your function **sum** should return 42.

sum.js contains a starting point which you should use. Download sum.js here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/js_sum/sum.js .
```

test.js contains code which will given will call your function and print the result. Download test.js here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/js_sum/test.js .
```

For example:

```
$ 2041 node test.js 1 2 3
node[11492]: pthread_create: Resource temporarily unavailable

<--- Last few GCs --->


<--- JS stacktrace --->

==== JS stack trace =========================================

Security context: 0x263961d9 <JSObject>
    2: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
[bytecode=0x2639cedd offset=378](this=0x4ea84101 <null>,process=0x5
27867a1 <process map = 0x2d3081a1>)

==== Details ================================================

[2]: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
 [bytecode=0x2639cedd offset=378](this=0x4ea84101 <null>,process=0x
527867a1 <process map = 0x2d3081a1>...

FATAL ERROR: Zone Allocation failed - process out of memory
 1: node::Abort() [node]
 2: 0x84b9e46 [node]
 3: v8::Utils::ReportOOMFailure(char const*, bool) [node]
 4: v8::internal::V8::FatalProcessOutOfMemory(char const*, bool) [n
```

You can assume all strings in the list are valid positive integers.

Your answer must be Javascript only. You can not use other languages such as Shell, Perl, Python or C.

You may not run external programs, e.g. via child_process or spawn.

No error checking is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 2041 autotest js_sum
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test11_js_sum sum.js
```

Sample solution for `sum.js`

```
// a functional programmming style solution

function sum(list) {
  // the unary + forces conversion to an integer (parseInt could also be used)
  return list.reduce((a, b) => a + +b, 0);
}

module.exports = sum;
```

Alternative solution for `sum.js`

```
// an iterative solution

function sum(list) {
  var total = 0;

  for (var i = 0; i < list.length; i++) {
    // the unary + forces conversion to an integer (parseInt could also be used)
    total += +list[i];
  }

  return total;
}

module.exports = sum;
```

Counting A Whale Species in Javascript

Your task is to write a Javascript function **species_count** which takes as its first argument a whale species and its second argument a list of whale observations, and counts the number of whales of that species in the list.
The list will be in this format:

```
[
    {
        "date": "09/09/18",
        "how_many": 11,
        "species": "Orca"
    },
    {
        "date": "04/11/17",
        "how_many": 41,
        "species": "Long-finned pilot whale"
    },
    {
        "date": "17/03/18",
        "how_many": 30,
        "species": "Southern right whale"
    },
    {
        "date": "17/08/18",
        "how_many": 31,
        "species": "Orca"
    },
]
```

If **species_count** was given "Orca" as its first argument and the above list as its second argument , it should return 42.
species_count.js contains a starting point which you should use. Download species_count.js here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/js_species_count/species_count.js .
```

test.js contains code which will given a whale species, and a JSON file containing a list of whale observations will read the JSON, call your function and print the result. Download test.js here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/js_species_count/test.js .
```

Note your Javascript function **species_count** does not have to read a file. **test.js** reads the file and passes the list it reads as **species_count**'s second argument.

whales.json contains an example list of whale observations. Download whales.json here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/js_species_count/whales.json .
```

Here is how to test your function using **test.js** and **whales.json**

```
$ 2041 node test.js Orca whales.json
node[11512]: pthread_create: Resource temporarily unavailable

<--- Last few GCs --->


<--- JS stacktrace --->

==== JS stack trace =========================================

Security context: 0x5a3161d9 <JSObject>
    2: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
[bytecode=0x5a31cecd offset=378](this=0x47684101 <null>,process=0x4
df867a1 <process map = 0x5f0081a1>)

==== Details ================================================

[2]: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
 [bytecode=0x5a31cecd offset=378](this=0x47684101 <null>,process=0x
4df867a1 <process map = 0x5f0081a1>...

FATAL ERROR: Zone Allocation failed - process out of memory
 1: node::Abort() [node]
 2: 0x84b9e46 [node]
 3: v8::Utils::ReportOOMFailure(char const*, bool) [node]
 4: v8::internal::V8::FatalProcessOutOfMemory(char const*, bool) [n
```

Your function does not have to handle differences in case or white-space.

Your answer must be Javsacript only. You can not use other languages such as Shell, Perl, Python or C.

You may not run external programs, e.g. via child_process or spawn.

No error checking is necessary.

When you think your program is working you can `autotest` to run some simple automated tests:

```
$ 2041 autotest js_species_count
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test11_js_species_count species_count.js
```

Sample solution for `species_count.js`

```
// a functional programmming style solution

function species_count(target_species, whale_list) {
  const target_whale_observations = whale_list.filter(observation => observation.species == target_species);
  const target_whale_counts = target_whale_observations.map(observation => observation.how_many);
  const target_whale_total = target_whale_counts.reduce((a, b) => a + b, 0);
  return target_whale_total;
}

module.exports = species_count;
```

Alternative solution for `species_count.js`

```
// terse functional programmming style solution

function species_count(target_species, whale_list) {
  return whale_list.map(
    observation => observation.species == target_species ? observation.how_many : 0
  ).reduce(
    (a, b) => a + b, 0
  );
}

module.exports = species_count;
```

Alternative solution for `species_count.js`

```
// an iterative solution

function species_count(target_species, whale_list) {
  var n_whales = 0;

  for (var i = 0; i < whale_list.length; i++) {
    if (whale_list[i].species == target_species) {
      // The + ensures conversion to an intger
      n_whales += +whale_list[i].how_many;
    }
  }

  return n_whales;
}

module.exports = species_count;
```

Counting A Whale Species in Perl

Your task is to write a Perl program **json_count.pl** which takes as its first argument a whale species and its second argument a file containing a list of whale observations, and prints the number of whales of that species in the file.
The file will be in this format:

```
[
    {
        "date": "09/09/18",
        "how_many": 11,
        "species": "Orca"
    },
    {
        "date": "04/11/17",
        "how_many": 41,
        "species": "Long-finned pilot whale"
    },
    {
        "date": "17/03/18",
        "how_many": 30,
        "species": "Southern right whale"
    },
    {
        "date": "17/08/18",
        "how_many": 31,
        "species": "Orca"
    },
]
```

If **json_count.pl** was given "Orca" as its first argument and the above file as its second argument , it should print 42.
For example:

```
$ ./json_count.pl Orca whales.json
117
$ ./json_count.pl 'Striped dolphin' whales.json
19
$ ./json_count.pl 'Southern right whale' whales.json
64
$ ./json_count.pl 'Whale Shark' whales.json
0
```

You can assume the formating of the file is the same as the example above. Your program does not have to handle the many other ways a JSON file might be formatted.

You can assume the "date", "how_many" and "species" fields in whale observations are always in the same order as the above example.

Your answer must be Perl only. You can not use other languages such as Shell, Python or C.

You may not run external programs, e.g. via system or backquotes.

You may use any Perl module installed on CSE systems.

No error checking is necessary.

When you think your program is working you can **autotest** to run some simple automated tests:

```
$ 2041 autotest json_count
```

When you are finished working on this exercise you must submit your work by running **give**:

```
$ give cs2041 test11_json_count json_count.pl
```

Sample solution for **json_count.pl**

```
#!/usr/bin/perl -w

# Count  observations of a whale species in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise

die "Usage: $0 <whale species> <file>\n" if @ARGV != 2;

$target_species = $ARGV[0];
$filename = $ARGV[1];

my $n_whales = 0;

open my $f, $filename or die "Can not open $filename\n";

while ($line = <$f>) {
    if ($line =~ /"how_many": (\d+)/i) {
        $how_many = $1;
    } elsif ($line =~ /"species": "(.*)"/i) {
        my $species = $1;
        if ($species eq $target_species) {
            $n_whales += $how_many;
        }
    }
}

close $f;

print "$n_whales\n";
```

Alternative solution for **json_count.pl**

```
#!/usr/bin/perl -w

#Count  observations of a whale species in  1 or more files
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# terse less readable version

die "Usage: $0 <whale species> <files>\n" if @ARGV != 2;

$target_species = shift @ARGV;
while (<>) {
    $how_many = $1 if /"how_many": (\d+)/i;
    $n_whales += $how_many if /"species": "$target_species"/i;
}
print "$n_whales\n";
```

Submission

When you are finished each exercise make sure you submit your work by running **give**.

You can run **give** multiple times. Only your last submission will be marked.

Don't submit any exercises you haven't attempted.

If you are working at home, you may find it more convenient to upload your work via give's web interface.

Remember you have until **Saturday 5 January 19:00** to complete this test.

Automarking will be run by the lecturer several days after the submission deadline for the test, using test cases that you haven't seen: different to the test cases `autotest` runs for you.

(Hint: do your own testing as well as running `autotest`)

After automarking is run by the lecturer you can view it here the resulting mark will also be available via via give's web interface