COMP[29]041 18s2

Week 12 Weekly Test

Sample Answers

COMP[29]041 18s2

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.

Total the Bill: Javascript

Your task is to write a Javascript function **total_bill** which calculates how much we have we have spent at the supermarket in the last week.
It will be given as its first argument a list of supermarket bills.

Each supermarket bill will be a list of objects.

Each object will have **name** and **price** fields.

Your Javascript function **total_bill** should return as a number the sum of all the prices.

For example if **total_bill** is given as argument:

```
[
  [
     {"name": "Rice", "price": "$4.29"},
     {"name": "Butter", "price": "$4.00"}
  ],
  [
     {"name": "Cheese", "price": "$8.82"},
     {"name": "Rice", "price": "$6.84"},
     {"name": "Pasta", "price": "$7.87"}
  ],
  [
     {"name": "Peanut Butter", "price": "$6.93"},
     {"name": "Bread", "price": "$3.32"},
     {"name": "Noodles", "price": "$4.71"}
  ],
  [
     {"name": "Cheese", "price": "$4.60"},
     {"name": "Rice", "price": "$8.23"},
     {"name": "Bread", "price": "$5.99"},
     {"name": "Pasta", "price": "$0.11"},
     {"name": "Ramen", "price": "$0.98"}
  ],
  [
     {"name": "Noodles", "price": "$0.38"}
  ]
]
```

it should return **67.07**

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

```
$ cp -n /web/cs2041/18s2/activities/js_total_bill/total_bill.js .
```

test.js contains code which will given a a JSON file containing a list of bills 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_total_bill/test.js .
```

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

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

```
$ cp -n /web/cs2041/18s2/activities/js_total_bill/bills.json .
```

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

```
$ 2041 node test.js bills.json
node[11549]: pthread_create: Resource temporarily unavailable

<--- Last few GCs --->


<--- JS stacktrace --->

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

Security context: 0x43a161d9 <JSObject>
    2: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
[bytecode=0x43a1cecd offset=378](this=0x46604101 <null>,process=0x2
5b867a1 <process map = 0x3ed881a1>)

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

[2]: /* anonymous */(aka /* anonymous */) [bootstrap_node.js:612]
 [bytecode=0x43a1cecd offset=378](this=0x46604101 <null>,process=0x
25b867a1 <process map = 0x3ed881a1>...

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
```

Javascript hint: **"Andrew".slice(2) == "drew"**

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

Your function does not have to handling rounding errors.

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_total_bill
```

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

```
$ give cs2041 test12_js_total_bill total_bill.js
```

Sample solution for `total_bill.js`

```
// a functional programmming style solution

function total_bill(bill_list) {
  const flattened_bill_list = bill_list.reduce((a, b) => a.concat(b), []);
  const prices = flattened_bill_list.map(item => item.price.slice(1));
  const total = prices.reduce((a, b) => a + +b, 0);
  return total;
}

module.exports = total_bill;
```

Alternative solution for `total_bill.js`

```
// a (too) terse functional programmming style solution

function total_bill(bill_list) {
  return bill_list.reduce((a, b) => a + b.map(item => item.price.slice(1)).reduce((a, b) => a + +b, 0), 0)
}

module.exports = total_bill;
```

Alternative solution for `total_bill.js`

```
// an iterative solution

function total_bill(bill_list) {
  var total = 0;

  for (var i = 0; i < bill_list.length; i++) {
    for (var j = 0; j < bill_list[i].length; j++) {
      total += +bill_list[i][j].price.slice(1);
    }
  }

  return total;
}

module.exports = total_bill;
```

Total the Bill: Perl

Your task is to write a Perl program **total_bill.pl** which calculates how much we have we have spent at the supermarket in the last week.
It will be given as its first argument a file containing a list of supermarket bills.

The file will be in this format:

```
[
  [
     {"name": "Rice", "price": "$4.29"},
     {"name": "Butter", "price": "$4.00"}
  ],
  [
     {"name": "Cheese", "price": "$8.82"},
     {"name": "Rice", "price": "$6.84"},
     {"name": "Pasta", "price": "$7.87"}
  ],
  [
     {"name": "Peanut Butter", "price": "$6.93"},
     {"name": "Bread", "price": "$3.32"},
     {"name": "Noodles", "price": "$4.71"}
  ],
  [
     {"name": "Cheese", "price": "$4.60"},
     {"name": "Rice", "price": "$8.23"},
     {"name": "Bread", "price": "$5.99"},
     {"name": "Pasta", "price": "$0.54"},
     {"name": "Ramen", "price": "$0.98"}
  ],
  [
     {"name": "Noodles", "price": "$0.38"}
  ]
]
```

If **total_bill.pl** was given this file as its first argument it should print **$67.50**
bills.json contains an example list of bills Download bills.json here, or copy it to your CSE account using the following command:

```
$ cp -n /web/cs2041/18s2/activities/perl_total_bill/bills.json .
```

Here is how **total_bill.pl** should behave:

```
$ total_bill.pl bills.json
$67.50
$ cat no_bills.json
[
]
$ total_bill.pl no_bills.json
$0.00
```

You must match the output format exactly.
Your function does not have to handle differences in case or white-space.

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 "name" and "price" fields 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 perl_total_bill
```

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

```
$ give cs2041 test12_perl_total_bill total_bill.pl
```

Sample solution for `total_bill.pl`

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

# Tally supermarket bills in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise

die "Usage: $0 <file>\n" if @ARGV != 1;

$filename = $ARGV[0];

my $total = 0;

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

while ($line = <$f>) {
    if ($line =~ /"price"\s*:\s*"\$(\d+\.\d+)/i) {
        $total += $1;
    }
}

close $f;

printf "\$%.2f\n", $total;
```

Alternative solution for `total_bill.pl`

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

# Tally supermarket bills in a file in  1 or more files
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# terse less readable version

/"price"\s*:\s*"\$(\d+\.\d+)/i and $total += $1 while <>;
printf "\$%.2f\n", $total;
```

What have We been Eating

Your task is to write a Shell script **eating.sh** which prints a list of the food we have bought at the supermarket in the last week. It will be given as its first argument a file containing a list of supermarket bills.

The file will be in this format:

```
[
  [
     {"name": "Rice", "price": "$4.29"},
     {"name": "Butter", "price": "$4.00"}
  ],
  [
     {"name": "Cheese", "price": "$8.82"},
     {"name": "Rice", "price": "$6.84"},
     {"name": "Pasta", "price": "$7.87"}
  ],
  [
     {"name": "Peanut Butter", "price": "$6.93"},
     {"name": "Bread", "price": "$3.32"},
     {"name": "Noodles", "price": "$4.71"}
  ],
  [
     {"name": "Cheese", "price": "$4.60"},
     {"name": "Rice", "price": "$8.23"},
     {"name": "Bread", "price": "$5.99"},
     {"name": "Pasta", "price": "$0.54"},
     {"name": "Ramen", "price": "$0.98"}
  ],
  [
     {"name": "Noodles", "price": "$0.38"}
  ]
]
```

If **eating.sh** was given this file as its first argument it should print

```
Bread
Butter
Cheese
Noodles
Pasta
Peanut Butter
Ramen
Rice
```

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

```
$ cp -n /web/cs2041/18s2/activities/eating/bills.json .
```

Here is how **eating.sh** should behave:

```
$ eating.sh bills.json
Bread
Butter
Cheese
Noodles
Pasta
Peanut Butter
Ramen
Rice
```

You must print the food in alphabetic order.
Each type of food must be printed only once.

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

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 "name" and "price" fields are always in the same order as the above example.

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

No error checking is necessary.

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

```
$ 2041 autotest eating
```

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

```
$ give cs2041 test12_eating eating.sh
```

Sample solution for `eating.sh`

```
#!/bin/sh

# Tally supermarket bills in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# Lines in file have this format
# {"name": "Peanut Butter", "price": "$6.93"},

egrep '"name"' "$1"|
cut -d\" -f4|
sort|
uniq
```

Alternative solution for `eating.sh`

```
#!/bin/sh

# Tally supermarket bills in a file
# andrewt@unsw.edu.au for a COMP[29]041 exercise
# Lines in file have this format
# {"name": "Peanut Butter", "price": "$6.93"},

egrep '"name"' "$1"|
sed '
    s/.*"name": "//
    s/".*//
    '|
sort|
uniq
```

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

COMP[29]041 18s2: Software Construction is brought to you by
the School of Computer Science and Engineering at the University of New South Wales, Sydney.
For all enquiries, please email the class account at cs2041@cse.unsw.edu.au
CRICOS Provider 00098G