

Scripting Languages G22.3033-002 Summer 2008

Example solutions for hw03

Assigned Th 6/5/2008, due Fr 6/13 at 9pm. 50 points.

<http://www.cs.nyu.edu/courses/summer08/G22.3033-002/>

Instructions for example solutions

These are example solutions. Please keep in mind that often, there is not just one correct solution to a question. If you come up with different answers, then it may be that both your answers and these answers here are correct. Of course, these answers here may also contain mistakes. If you spot a mistake, please let me know so I can correct it.

Example solutions for Homework 3

solutions-hw03-1 Regular expressions

a.

```
(|0x|0X)
(a|b|c|d|e|f|A|B|C|D|E|F|0|1|2|3|4|5|6|7|8|9)
(a|b|c|d|e|f|A|B|C|D|E|F|0|1|2|3|4|5|6|7|8|9)*
```

b. $\wedge s^*(?:\S+\s+)^{3}(\S+)\s+(\S+)$

solutions-hw03-2 Static and dynamic scoping

a. The program prints `g f`. Variable `$a` in Function `g` uses dynamic scoping. Therefore, it is visible in the callee function `h`: since `g` is the closest calling function, it hides variable `$a` from the nesting function `f`. The second print statement in function `f`, on the other hand, does not see `$a` from function `g` anymore, because it has already returned, and is not a calling function anymore, let alone the closest one.

b. Consider this example:

```
#!/usr/bin/perl -w
sub g { $a = "g" }
g();
print $a, "\n";
sub f { print $a, "\n" }
f()
```

This code prints `g g`. That means that the assignment `$a = "g"` implicitly declared a fresh global variable `$a` that is visible both at the top level and in another function `f` that is neither nested in nor called by function `g`.

solutions-hw03-3 CSV to fixed-width (Perl)

a.

```
0 0 10 0
10 0 10 0
10 -10 10 0
6.666666666666667 -3.333333333333333 10 0
```

- b. (Look at the manual pages!)
- c. The `while (<>)` loop header reads lines from standard input to `$_`. The function `chomp` trims whitespace away from `$_`. The function `split` divides `$_` up into comma-separated values to initialize an array reference to store in `@rows`.
- d. Variable `$row` holds a reference to an array, and `@$row` dereferences it to get the array. Since it gets used like a scalar, Perl interprets it as the length of the referred-to array.

solutions-hw03-4 Average column (Perl)

```
#!/usr/bin/perl -w
while (<>) {
    chomp;
    push @rows, [ split /\s/ ];
}
#----- added code for hw03-4 start -----
for $row (@rows) {
    for ($i = 0; $i < @$row; $i++) {
        $sums[$i] += $row->[$i];
        $count[$i]++;
    }
}
for ($i = 0; $i < @sums; $i++) {
    $avg[$i] = $sums[$i] / $count[$i];
}
push @rows, [ @avg ];
#----- added code for hw03-4 end -----
for $row (@rows) {
    for ($i = 0; $i < @$row; $i++) {
        $w = length $row->[$i];
        $widths[$i] = $w if !$widths[$i] || $w > $widths[$i];
    }
}
for $row (@rows) {
    for ($i = 0; $i < @$row; $i++) {
        print " " x (1 + $widths[$i] - length $row->[$i]);
        print $row->[$i];
    }
    print "\n";
}
```

solutions-hw03-5 Euclidian distance (Perl)

```
#!/usr/bin/perl -w
while (<>) {
    chomp;
    push @rows, [ split /\s/ ];
}
#----- added code for hw03-5 start -----
```

```

for ($r = 0; $r < @rows; $r++) {
    $delta = 0;
    for ($i = 0; $i < @{$rows[$r]}; $i++) {
        $delta += ($rows[$r]->[$i] - $rows[0]->[$i]) ** 2;
    }
    push @{$rows[$r]}, $delta ** .5;
}
#----- added code for hw03-5 end -----
for $row (@rows) {
    for ($i = 0; $i < @$row; $i++) {
        $w = length $row->[$i];
        $widths[$i] = $w if !$widths[$i] || $w > $widths[$i];
    }
}
for $row (@rows) {
    for ($i = 0; $i < @$row; $i++) {
        print " " x (1 + $widths[$i] - length $row->[$i]);
        print $row->[$i];
    }
    print "\n";
}

```