

Scripting Languages G22.3033-002 Summer 2008

Example solutions for hw04

Assigned Th 6/12/2008, due Fr 6/20 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 4

solutions-hw04-1 Weak, strong, static, and dynamic typing

- a. Weak typing means that there are many implicit type conversions. In other words, the same value will be interpreted differently depending on what type is expected. This increases flexibility, but may surprise the programmer and thus cause bugs.

Dynamic typing means that most type checks occur at runtime rather than at compile time. This is not the same as weak typing, because the dynamic checks need not be weak. For instance, Scheme has strong dynamic typing: checks are deferred until runtime, but conversions have to be explicit in most cases.

- b. I put VBA half-way between static and dynamic typing, because it has gradual typing, allowing the programmer to choose how many checks happen at compile time and how many at runtime. I put VBA half-way between weak and strong typing because there are not as many implicit conversions as, for example, in Perl.

solutions-hw04-2 Context in Perl

- a.

```
print "12 ", (isprime 12) ? "is" : "is not", " a prime\n";
print "prime factors of 12: ";
for $f (isprime 12) { print "$f " }
print "\n";
print "13 ", (isprime 13) ? "is" : "is not", " a prime\n";
print "prime factors of 13: ";
for $f (isprime 13) { print "$f " }
```

- b. The line `push @listout, &block() for (@listin);` makes heavy use of the default variable `$_`. The statement modifier `for(@listin)` iterates over the elements of `@listin`, putting each one in `$_`. Therefore, when the block gets called by `&block()`, the variable `$_` holds an array element. The code in the block can use this for computation.

- c.

```
@in = (1, 2, 3);
my @out = map_block { $_ + 5 } @in;
for (@out) { print $_, ' ' }
```

solutions-hw04-3 Bidirectional mapping (Perl)

```
use strict; use warnings;
package TwoWayMap;
sub new {
    my ($cls, %init) = @_;
    my $self = bless( { MAP => {}, REV => {} }, $cls );
    for my $key (keys %init) {
        $self->put($key, $init{$key});
    }
    return $self;
}
sub put {
    my ($self, $key, $val) = @_;
    $self->{MAP}{$key} = $val;
    $self->{REV}{$val} = $key;
}
sub get {
    my ($self, $key) = @_;
    return $self->{MAP}{$key};
}
sub rev {
    my ($self, $val) = @_;
    return $self->{REV}{$val};
}
1
```

solutions-hw04-4 Hierarchical clustering (Perl)

```
#!/usr/bin/perl
use strict; use warnings;

# hash, mapping from string to array reference
# key = cluster name, such as '(A:4:C)' or 'B'
# first array element = weight (number of clustered elements)
# remaining array elements = sum (of columns of all elements)
our %clusters = ();

sub sanity_print {
    print "\n";
    for my $key (keys %clusters) {
        my $weight = $clusters{$key}->[0];
        printf "%20s => %d x [" , $key, $weight;
        for (my $col = 1; $col < @{$clusters{$key}}; $col++) {
            print ", " unless $col == 1;
            print $clusters{$key}->[$col] / $weight;
        }
        print "]\n";
    }
}
```

```

    }
    print "}\n";
}

while(<>) {
    chomp;
    my @row = split /\s*,\s*/;
    my $key = $row[0];
    $row[0] = 1; # weight 1 = cluster contains one row
    $clusters{$key} = [ @row ];
}

sub euclid_distance {
    my ($cluster1, $cluster2) = @_;
    my ($weight1, $weight2) = ($$cluster1[0], $$cluster2[0]);
    my $result = 0;
    for (my $i = 1; $i < @$cluster1; $i++) {
        my ($c1, $c2) = ($$cluster1[$i], $$cluster2[$i]);
        $result += ($c1 / $weight1 - $c2 / $weight2) ** 2.0;
    }
    $result **= 0.5;
    return $result;
}

sub find_closest_pair {
    my %cls = @_;
    my @keys = keys %cls;
    my ($res1, $res2) = ($keys[0], $keys[1]);
    my $min_dist = euclid_distance($cls{$res1}, $cls{$res2});
    for my $i (@keys) {
        for my $j (@keys) {
            if ($i ne $j) {
                my $dist = euclid_distance($cls{$i}, $cls{$j});
                if ($dist < $min_dist) {
                    $res1 = $i;
                    $res2 = $j;
                    $min_dist = $dist;
                }
            }
        }
    }
    return ($min_dist, $res1, $res2);
}

sub combine_clusters {
    my ($cluster1, $cluster2) = @_;
    my @result = ();

```

```
    for (my $i = 0; $i < @$cluster1; $i++) {
        push @result, $$cluster1[$i] + $$cluster2[$i];
    }
    return [ @result ];
}

while (1 < scalar(keys %clusters)) {
    #sanity_print(); # uncomment this line to test/debug
    my ($dist, $k1, $k2) = find_closest_pair(%clusters);
    my $new_key = "($k1:$dist:$k2)";
    my $new_val = combine_clusters($clusters{$k1}, $clusters{$k2});
    delete $clusters{$k1};
    delete $clusters{$k2};
    $clusters{$new_key} = $new_val;
}

print keys %clusters, "\n";
```