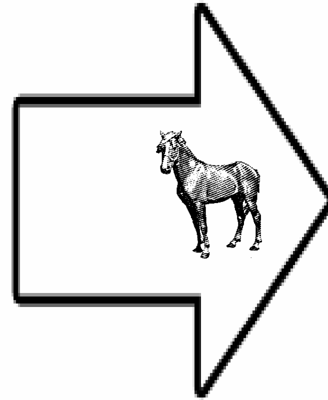
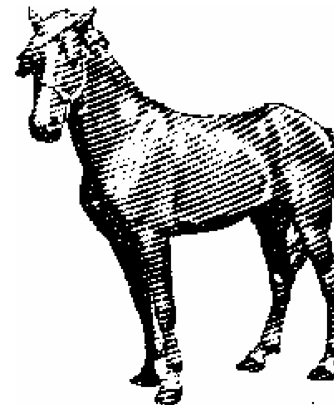
1.1.2.8.1

Intermediate Perl
Session 1

- references
- complex data structres

References





- references are scalars which point to variables
  - they hold the variable's memory location
  - value of the variable can be obtained by dereferencing
- size of scalar variable is always the same size, regardless of type of variable
  - easy to pass around a reference, instead of large variable

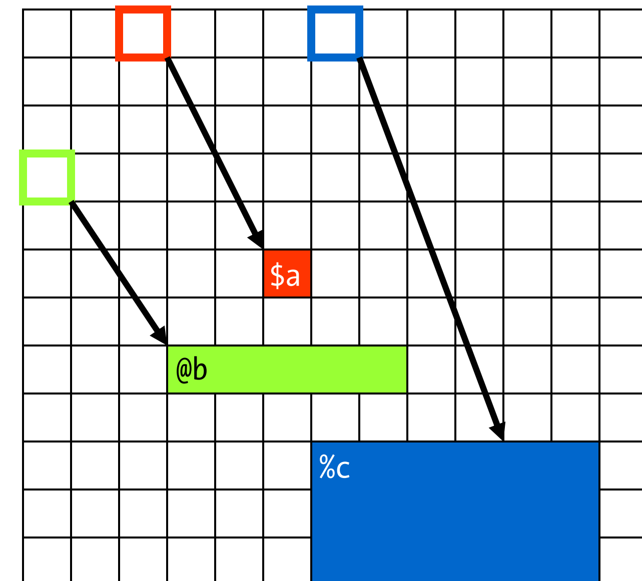
References Point to Memory Addresses

- perl's basic variable types are $scalars, @arrays, and %hashes
- each variable may occupy different amounts of memory

```
$a = 10
@b = (1,2,3)
%c = (one=>1,two=>2)
```

- each variable can have a reference

```
$a_ref = \$a
$b_ref = \@b
$c_ref = \%c
```



Creating References

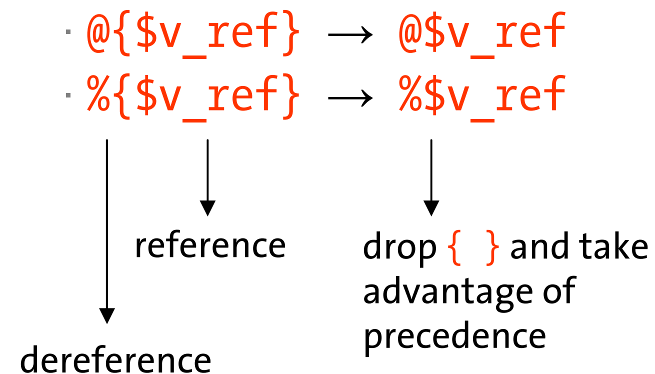
	scalar $x	array @a	hash %h
reference	`$x_ref = \$x`	`$a_ref = \@a`	`$h_ref = \%h`
dereference	`$$x_ref`	`@$a_ref`	`%$h_ref`

- to create a reference add \ at the front of the variable
- to dereference, add the appropriate variable prefix to the reference
  - e.g. add @ if dereferencing an array reference
- we'll leave scalar references for now, since they're least used

Rule #1

- let $v_ref be a reference to variable
- you can replace variable by {$v_ref} in any code

```
@a = (1,2,3);
$a_ref = \@a;

%h = (one=>1,two=>2);
$h_ref = \%h;

print @a
print @{$a_ref}

print %h
print %{$h_ref}
```



References as Strings and Ref()

- If you try to print a reference variable, you get a text string
  - the string is useless, except for debugging
  - it indicates the referent type
  - it indicates the memory address

```
my $x = 10;
my @a = (1,2,3);
my %h = (one=>1,two=>2);

my $x_ref = \$x;
my $a_ref = \@a;
my $h_ref = \%h;

print $x;        10
print $x_ref;    SCALAR(0x80cb34c)
print @a;        123
print $a_ref;    ARRAY(0x80cb358)
print %h;        two2one1
print $h_ref;    HASH(0x80cf3d4)
```

Identifying References with Ref()

- the function ref() is used to identify references
  - ref($var) returns undef if $var is not a reference
  - ref($var) returns the string "SCALAR", "ARRAY", or "HASH"

```
print ref($x);
print ref($x_ref);       SCALAR
print ref($a_ref);       ARRAY
print ref($h_ref);       HASH

if (ref($mystery_variable) eq "ARRAY") {
  print @$mystery_variable;
} else {
  print "not an array reference";
}
```

Anonymous References – Part 1

- previously, we needed a pre-existing variable to create a reference.

- suppose I want a reference to the list (1,2,3)

```
@a = (1,2,3)
$a_ref = \@a;
```

- once the reference is created, @a is no longer needed
  - (1,2,3) can be accessed by @$a_ref

- an anonymous reference obviates the need for a named variable

```
$a_ref = [1,2,3];
```

Anonymous References – Part 2

- anonymous references are going to initially annoy you because they add yet another set of bracket rules
  - train your eyes
  - keep patient
  - collect rewards

	array `@a = (1,2,3)`	hash `%h = (one=>1,two=>2)`
reference created from variable	`$a_ref = \@a`	`$h_ref = \%h`
anonymous reference	`$a_ref = [1,2,3]`	`$h_ref = {one=>1,two=>2}`

Anonymous References – Part 3

- to recover the variable refered by an anonymous reference – dereference!
  - remember, there is no associated named variable

```
@a   = (1,2,3);
$ar1 = \@a;       reference points to named variable
$ar2 = [1,2,3];   anonymous reference does not point to any named variable

%h   = (one=>1,two=>2)
$hr1 = \%h
$hr2 = {one=>1,two=>2};
```

Perl Tattoo

@
list
`@a=(1,2,3)`

%
hash
`%h=(one=>1,two=>2)`

\@
list reference
`$ar = \@a`

\%
hash reference
`$hr = \@h`

( )
list or hash

[ ]
anonymous list reference
`$ar = [1,2,3]`

{ }
anonymous hash reference
`$hr = {one=>1,two=>2}`

References for Complex Data Structures

- to create complex data structures (e.g. lists of lists) you need references because
  - list elements must be scalars
  - hash values must be scalars

- remember that lists collapse
  - ((1,2),(3,4)) not a two-element list of lists but a 4-element list of scalars

```
# these are the same
@a = ( (1,2),(3,4) )
@b = (1,2,3,4)
```

- a list of lists is created by making a list of references to lists

Dereferencing – Lists

```
$a_ref = [1,2,3]
```

- to dereference `$a_ref`

"go blind" method

```
${$a_ref}[0]
${$a_ref}[1]
${$a_ref}[2]
```

"stay sane" method

```
$a_ref->[0]
$a_ref->[1]
$a_ref->[2]
```

```
              $a[0]
compare with  $a[1]  for array @a
              $a[2]
```

Dereferencing – Hashes

```
$h_ref = {one=>1,two=>2}
```

- to dereference $h_ref

"go blind" method

```
${$h_ref}{one}
${$h_ref}{two}
```

"stay sane" method

```
$h_ref->{one}
$h_ref->{two}
```

```
                $h{one}
compare with    $h{two}    for hash %h
                $h{three}
```

Reference table to references

	array @a = (1,2,3)	hash %h = (one=>1,two=>2)
reference created from variable	$a_ref = \@a	$h_ref = \%h
anonymous reference	$a_ref = [1,2,3]	$h_ref = {one=>1,two=>2}
dereferencing	$a_ref->[0] $a_ref->[1] $a_ref->[2]	$h_ref->{one} $h_ref->{two}

Deeply Nested Structures

- consider the following hash, whose values are list references
  - $h is a reference to a hash of lists

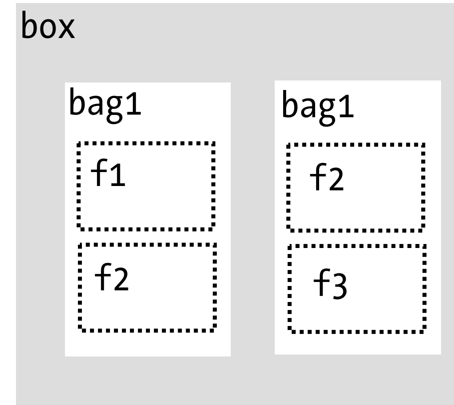
```
$even = [2,3,4];
$odd  = [1,3,5];
$h    = {even=>$even,odd=>$odd};
```

- $h->{even} is the value of the "even" key which is $even, a list reference

```
$h                 reference to hash
$h->{even}         reference to value of good

$h->{even}->[0]    first element in value of even (2)
$h->{even}[0]      first element in value of even (2)

$h->{odd}[0]       first element in value of odd (1)
$h->{odd}[1]       second element in value of odd (3)
$h->{odd}[2]       third element in value of odd (5)
```

More Deeply Nested Structures

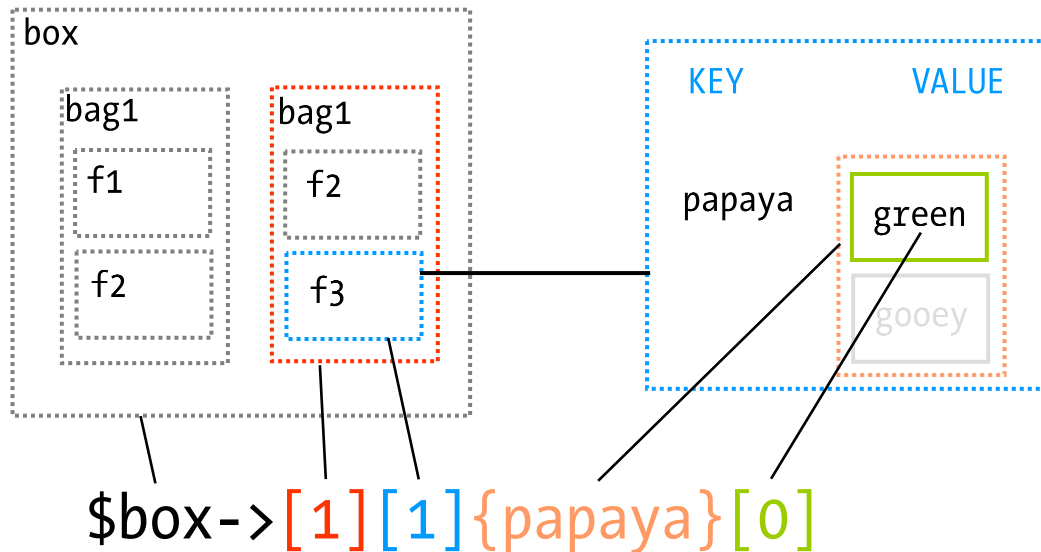
```
$f1     = { apple  => [qw(red tasty)] }
$f2     = { banana => [qw(yellow squishy)] }
$f3     = { papaya => [qw(green gooey)] }

$bag1   = [$f1,$f2];
$bag2   = [$f2,$f3];

$box    = [$bag1,$bag2];
```



`$f1->{apple}`	ARRAY(0x) (reference to anonymous list)
`$f1->{apple}[0]`	red
`$bag1->[0]`	HASH(0x) (reference to $f1)
`$bag1->[0]{apple}`	ARRAY(0x) (reference to anonymous list)
`$bag1->[0]{apple}[1]`	tasty
`$box->[1][1]{papaya}[0]`	green
`${${${${$box}[1]}[1]}{papaya}}[0]`	I'm blind

More Deeply Nested Structures

```
$f1     = { apple  => [qw(red tasty)] }
$f2     = { banana => [qw(yellow squishy)] }
$f3     = { papaya => [qw(green gooey)] }

$bag1   = [$f1,$f2];
$bag2   = [$f2,$f3];

$box    = [$bag1,$bag2];
```



Iterating through the Box of Bags of Fruits

```
$f1     = { apple  => [qw(red tasty)] };
$f2     = { banana => [qw(yellow squishy)] };
$f3     = { papaya => [qw(green gooey)] };

$bag1   = [$f1,$f2];
$bag2   = [$f2,$f3];

$box    = [$bag1,$bag2];

for $bag (@$box) {
  # each $bag is a list reference
  for $fruit (@$bag) {
    # each $fruit is a hash reference
    for $fruit_name (keys %$fruit) {
      # each $fruit_prop is a list reference
      for $fruit_prop (@{$fruit->{$fruit_name}}) {
            print “$fruit_name is $fruit_prop”;
      }
    }
  }
}
```

1.1.2.8.1

Intermediate Perl Session 1

- [ ] is for array lookup
- [ ] is for anonymous arrays

- { } is for hash key lookup
- { } is for anonymous hashes