Hashes

Sofia Robb

First a few words about when to use: Arrays and Hashes

- Use an array when ordered values matter or when you have an arbitrary list of values.
    - Sort list:

(0.001, 0.01 , 0.1 , 1)

('apple', 'bee', 'cedar' , 'deer')

- - States:

('MD','UT','CA','MO','OR','NM');

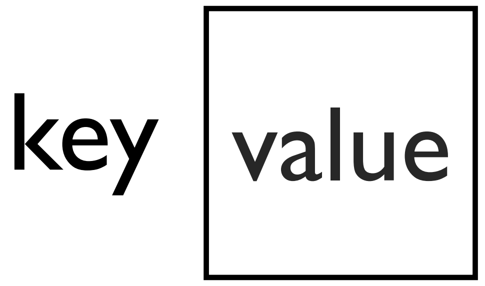
- Use a hash when order does not matter but you have paired information
    - dictionaries : Word => Definition
    - FASTA : Gene => Sequence

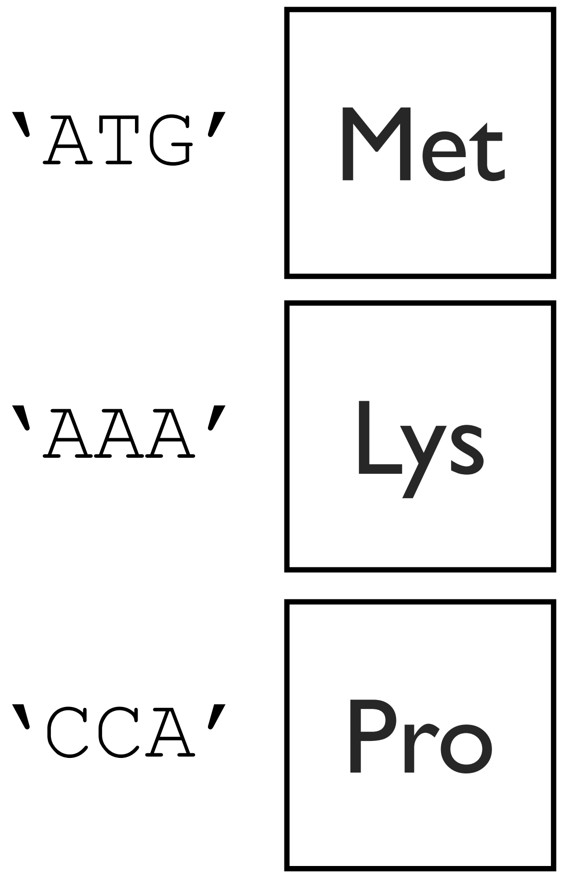
Hashes

- Perl hashes are denoted with a ‘%’ symbol like this: %data
- Each key and each value contains a scalar value for example this could be
  - a number
  - a letter
  - a word
  - a sentence
  - a scalar variable like $scalar_variable
  - a gene ID
  - a sequence

What is a hash?

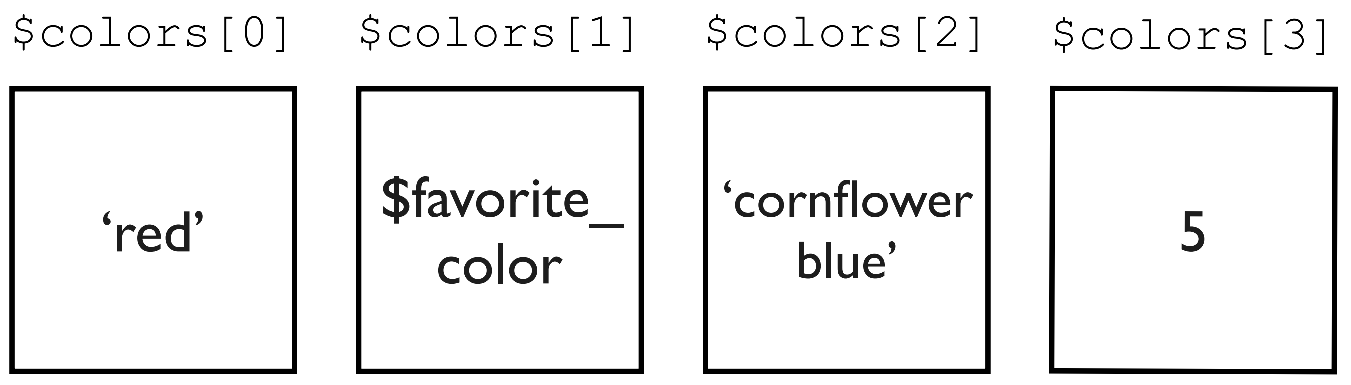
- A hash is an associative array made up of key/value pairs.
- Like a dictionary
- And unlike an array a hash is <u>unordered</u>.
- Keys need to be Unique!!



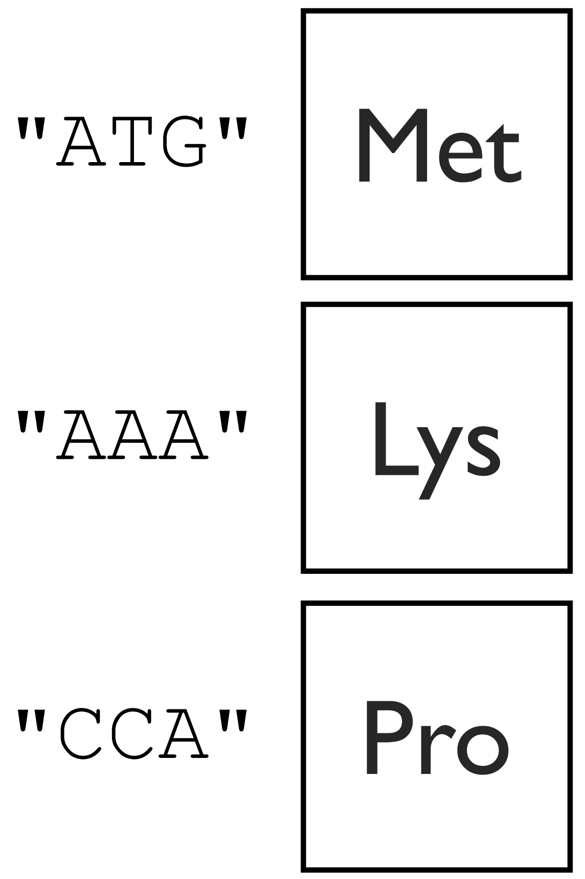


A key is like a descriptive array index.

An array



A hash



The array index [0] is similar to the key "ATG".

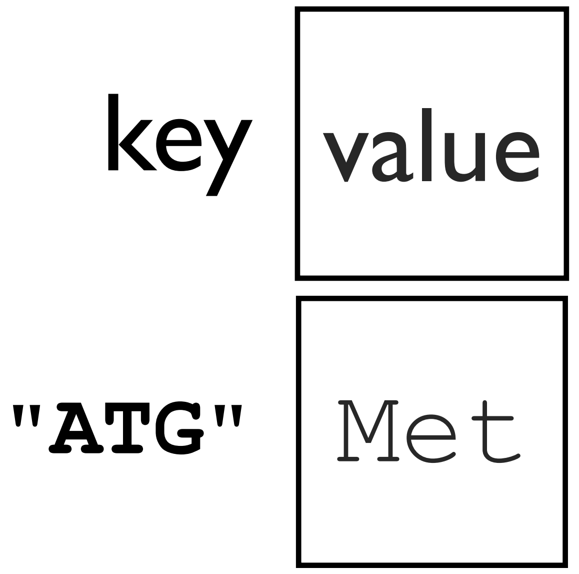
The key "ATG" is used to access the value "Met", just as [0] is used to access "red"

But the key/value pairs are not stored in order

Creating a Hash *All At Once*

The hash %genetic_code is built with key/value pairs

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```



Creating a Single Hash *One Key/Value Pair at a time.*

```
$hash{key} = “value”;
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

SAME AS:

```
$genetic_code{"ATG"} = "Met";
$genetic_code{"AAA"} = "Lys";
$genetic_code{"CCA"} = "Pro";
```

Even though there are 3 lines, this is still only 1 hash

<u>Accessing</u> a hash value using a key

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

Look!! GET/SET are mirror images!!

```
SET: $genetic_code{"ATG"} = $value;
GET: $value = $genetic_code{"ATG"};
```

```
my $aa = $genetic_code{"ATG"};
print "ATG translates to $aa\n";
```

ATG translates to Met

Each value of the hash is a scalar therefore we use the '$' when we refer to an individual value.

Hash keys are surrounded by squiggly brackets {}

keys() returns an unordered list of the keys of a hash

```
@array_of_keys = keys (%hash);
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

```
my @codons = keys (%genetic_code);
print join("--",@codons), "\n";
CCA--AAA--ATG
```

values() returns an unordered list of values

```
@array_of_values = values(%hash);
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

You can use `sort values` to be sure that the order of the values is always the same.

```
my @amino_acids = values(%genetic_code);
print join("--",@amino_acids), "\n";
Pro--Lys--Met
```

Iterating through a hash by looping through an list of hash keys.

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

Remember: the key is used to access the value

`$value = $hash{$key}`

```
foreach my $codon (keys %genetic_code){
   my $aa = $genetic_code{$codon};
   print "$codon translates to $aa\n";
}
```

```
CCA translates to Pro
AAA translates to Lys
ATG translates to Met
```

Sorting and iterating through the keys of a hash

```
my %genetic_code = (
  "ATG" => "Met",
  "AAA" => "Lys",
  "CCA" => "Pro",
);
```

Remember: hash keys are unordered so we use `sort` to be sure that the order is always the same.

```
foreach my $codon (sort(keys %genetic_code)){
   my $aa = $genetic_code{$codon};
   print "$codon translates to $aa\n";
}
```

```
AAA translates to Lys
ATG translates to Met
CCA translates to Pro
```

Nested Functions:
(sort (keys %hash))

1. keys() returns a list of all the keys in %hash
2. This list is passed on to sort().
3. sort() will then proceed to sort the list

Iterating through a hash and sorting by the values

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

Remember: the key is used to access the value

```
$value = $hash{$key}
```

$a and $b are keys, the value of key $a is being sorted in comparison to the value of key $b

evaluates to a value — evaluates to a value

```
foreach my $codon (sort { $genetic_code{$a} cmp $genetic_code{$b} } keys %genetic_code){
   my $aa = $genetic_code{$codon};
   print "$codon translates to $aa\n";
}
```

```
AAA translates to Lys
ATG translates to Met
CCA translates to Pro
```

we can create a custom sort function using {$a cmp $b}

Adding additional key/value pairs

```
$hash{key} = “value”;
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

```
$genetic_code{"TGT"} = "Cys";
```

```
foreach my $codon (keys %genetic_code){
  print "$codon -- $genetic_code{$codon}\n";
}
```

```
CCA -- Pro
AAA -- Lys
ATG -- Met
TGT -- Cys
```

Over-writing values

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

```
$hash{key} = “value”;
```

```
print "Before/ATG: " , $genetic_code{"ATG"} ,"\n";

$genetic_code{"ATG"} = "start_codon";

print "After/ATG: " , $genetic_code{"ATG"} ,"\n";
```

```
Before/ATG: Met
After/ATG: start_codon
```

Since keys need to be unique a value is over-written when using a preexisting key.

Deleting key/value pairs

```
delete $hash{"KEY"};
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

```
delete $genetic_code{"AAA"};

foreach my $codon (keys %genetic_code){
  print "$codon -- $genetic_code{$codon}\n";
}
CCA -- Pro
ATG -- Met
```

Use exists() to test if a key exists.

```
exists $hash{"KEY"}
```

```
my %genetic_code = (
 "ATG" => "Met",
 "AAA" => "Lys",
 "CCA" => "Pro",
);
```

key exists?	return value
yes	1
no	'' empty string is false

```
my $codon = "ATG";
if (exists $genetic_code{$codon}){
  print "$codon -- $genetic_code{$codon}\n";
}else{
  print "key: $codon does not exist\n";
}
ATG -- Met
##when $codon= "TTT", code prints "key: TTT does not exist"
```

Auto increment hash values

```
Auto increment scalars:
my $num = 1;
print $num , "\n";  #prints 1
$num++;              #same as $num=$num +1;
print $num , "\n";  #prints 2
```

```
Auto increment hash values:
my %hash;
$hash{books} = 0;
print $hash{books}, "\n";  #prints 0
$hash{books}++; #same as $hash{books} = $hash{books} + 1
print $hash{books} , "\n"; #prints 1
```

nothing + 1 equals 1

```
my %hash;

~~$hash{books} = 0;~~
~~print $hash{books}, "\n";~~

$hash{books}++;
print $hash{books} , "\n"; # prints 1
```

When we first start, the key ‘books’ doesn’t exist.
We try to add `1` to nothing, so the total is `1`.

Using hashes for keeping count

```
my $seq = "ATGGGCGTATGCAATT";
my @nucs = split "", $seq;
print "@nucs\n";
#A T G G G C G T A T G C A A T T

my %nt_count;
foreach my $nt (@nucs){
        $nt_count{$nt}++;
}

foreach my $nt (keys %nt_count){
        my $count = $nt_count{$nt};
        print "$nt\t$count\n";
}

A       4
T       5
C       2
G       5
```

Parse a tab-delimited file and store the file columns in a hash.

```
my $file = shift @ARGV;
open (INFILE, '<', $file)
  or die "can't open file $file $!\n";
my %hash;
while (my $line = <INFILE>){
        chomp $line;
        my ($key, $value) = split /\t/, $line;
        $hash{$key} = $value;
}
foreach my $key (sort keys %hash){
        my $value = $hash{$key};
        print "key:$key value:$value\n";
}
```

input:

genes.txt (tab delimited file):

```
geneA\tATGC
geneB\tTGCA
geneC\tAACT
```

output:

STDOUT (on the screen):

```
key:geneA value:ATGC
key:geneB value:TGCA
key:geneC value:AACT
```