

Foundations in Perl Programming – for experienced programmers

**Week Two**

# This weeks topics

---

- **Review assignments, walk-through your code**
- **More details on...**
  - **Scalars**
  - **Arrays**
  - **Hashes**
  - **Subroutines**
  - **Loops**
- **Quick prelude on the Debugger**
- **Regular Expressions**
- **Assignments**
- **Start thinking about your project!!! Your idea is due by next class.**

## Review Assignments / Code Walk-throughs

---

Approx  
50 mins

## More Details

---

### Topics

- Scalars
- Arrays
- Subroutines
- Loops

Manual: p35-54

Approx  
50 mins

## More Details

---

### Scalars

DEFINED/UNDEF

undef.pl

Testing state

Tf-vartest.pl

Manual: p36-40

## More Details

---

### Arrays

Pushing/Popping

push.pl

pop.pl

Shifting/Unshifting

shift.pl

unshift.pl

push2.pl

Manual: p41-43

## Hashes (Associative Arrays)

---

A quick visual...

```
@arr = (298, 114, 31, 12, 2, 67, 55);  
  
$arr[0] = 298;  
$arr[1] = 114;  
$arr[2] = 31;  
$arr[3] = 12;  
$arr[4] = 2;  
$arr[5] = 67;  
$arr[6] = 55;
```

```
%batter_stats =  
(  
  'Batting Average' = 298;  
  'Hits' = 114;  
  'Homeruns' = 31;  
  'Double' = 12;  
  'Triples' = 2;  
  'RBI' = 67;  
  'Runs' = 55;  
)
```

Manual: p78-84

Approx  
2-2.5 hours

## Hashes (Associative Arrays)

Creating

hash1.pl

hash2.pl

```
use strict;

# create an empty hash
my %hsh = ();

#add some elements
$hsh{'firstname'} = 'John';
$hsh{'middle_init'} = 'H.';
$hsh{'lastname'} = 'Smith';

print $hsh{lastname};
```

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

print $hsh{lastname};
```

## Hashes (Associative Arrays)

### Adding and element

hash3.pl

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my $title = "Supreme Commander";

$hsh{title} = $title;

print $hsh{firstname};
print " ";
print $hsh{middle_init};
print " ";
print $hsh{lastname};
print ", ";
print $hsh{title};
```

## Hashes (Associative Arrays)

---

### Deleting an element hash4.pl

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

delete $hsh{middle_init};

if (exists($hsh{middle_init}))
{
  print "key exists";
}
else
{
  print "key does not exist";
}
```

## Hashes (Associative Arrays)

---

Get keys

hash5.pl

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my $key="";
foreach $key (keys %hsh)
{
  print $key . "\n";
}
```

## Hashes (Associative Arrays)

---

Get values

hash6.pl

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my $val="";
foreach $val (values %hsh)
{
  print $val . "\n";
}
```

## Hashes (Associative Arrays)

### Finding a particular key

hash8.pl

```
my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

# exists is used to verify key
if (exists($hsh{middle_init}))
{
  print $hsh{middle_init};
  print "\n"
}
else
{
  print "key does not exist";
}
```

## Hashes (Associative Arrays)

### Sorting

hash10.pl

hash11.pl

#### By key

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my $key="";
foreach $key (sort keys %hsh)
{
  print $key . "\n";
}
```

#### By Value

```
use strict;

my %hsh =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my $val="";
foreach $val (sort values %hsh)
{
  print $val . "\n";
}
```

## Hashes (Associative Arrays)

### Merging

hash12.pl

```
use strict;
my %person =
(
  firstname => 'John',
  middle_init => 'H.',
  lastname => 'Smith',
);

my %phone =
(
  area_cde => '206',
  exchange => '466',
  extension => '1515',
);

my %person_phone = (%person, %phone);

my $key="";
my $val="";
while (($key, $val) = each(%person_phone))
{
  print "$key => $val\n";
}
```

## More Details

---

### Subroutines

Passing parameters ——— subb.pl sub2.pl

Handling multiple arguments ——— subd.pl

Returning values ——— subf.pl subg.pl subi.pl

Recursion ——— subh.pl getfiles1a.pl

Manual: p45-54

## More Details

---

### Loops

Last ———— looplast.pl

Next ———— loopnext.pl

Redo

# Quick Intro to Debugger

---

## Walk-through

- List source
- Stepping
- Working with variables
- Redisplaying and executing debugger commands
- Breakpoints
- Writing/Changing code in the debugger (pretty cool)
- Watching

Manual: p85-91

Approx  
30 mins

## Regular Expressions

---

- Frightening to first look at. Your inclination may be to deal with these issues by writing conventional code and not using regular expressions. Trust me, you **do not** want to do this. You want to learn how to use these, they are powerful beyond your wildest imagination (at this point in your learning Perl)
- There is a steep curve in learning how to use these. Unless you are very fortunate it is going to take you some time (and maybe some grief) to get on board. I will not tell you not to be frustrated, but I will tell you to be diligent and tenacious with this subject. I cannot imagine Perl without regular expressions (This subject kicked my butt for a long time, and detracted me for years from learning this language. Most unfortunate for me.)
- Do not feel that anyone thinks you are dumb for struggling with this subject. It is a very difficult subject and may require a new way of thinking about string manipulation on your part.
- Finally, and most important: You need to learn how to read regular expressions just like you learned how to read the written word. I will be stressing this as the most important theme throughout this subject.

Approx  
2+ hours

# Regular Expressions

---

## Topics to cover

A wee bit of matching theory (we can apply)

Quantifiers

Character classes

Parenthesis and Alternation

Dot operator

Anchors

Misc topics

Manual: p55-65,74,75

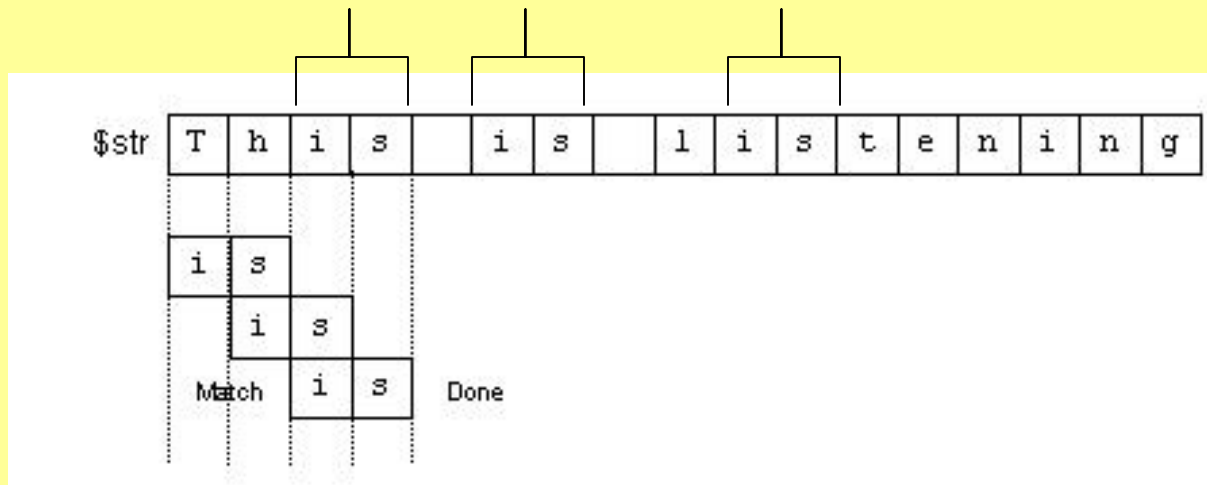
# Regular Expressions

## A little matching theory

**Note:** *some of what follows will be a review of last weeks materials.*

```
$str = "This is listening";  
$str =~ m/is/
```

Which "is" do we match?



**The first match wins.**



# Regular Expressions

---

## A little matching theory

### Matching constructs

- Literals
- Character classes
- Metacharacters

### Matching framework

```
m/ / or just / /      #matching  
s/ / /                #substitution
```

### Matching operators

```
=~      Matches  
!~      Does not match
```

# Regular Expressions

## Quantifiers

<b>?</b>	means that the character preceding is optional, though one is allowed.
<b>*</b>	means match zero or more of the preceding character, though none are required, any amount allowed.
<b>+</b>	means match one or more of the preceding character, one is required, any amount allowed.

```
my $str="flavour";
```

```
my $str="flavor";
```

```
$str =~ /flavou?r/)
```

```
regex1a.pl
```

**This reads**, "match all the literal characters "f", "l", "a", "v", "o", and may or may not include a "u", followed by literal "r".

# Regular Expressions

---

## Quantifiers

regex1b.pl

```
use strict;

my @arr=qw(feel felt fertile feat);
my $arr=@arr; # array size
my $i=0;

for ($i=0; $i<$arr; $i++)
{
    if (@arr[$i] =~ /fea*/)
    {
        print("@arr[$i]\n");
    }
}
```

**This reads?** \_\_\_\_\_

**Will Match?** \_\_\_\_\_  
\_\_\_\_\_

# Regular Expressions

---

## Quantifiers

regex1c.pl

```
use strict;

my @arr=qw(feel felt fertile feat);
my $arr=@arr; # array size
my $i=0;

for ($i=0; $i < $arr; $i++)
{
    if (@arr[$i] =~ /fea+/)
    {
        print("@arr[$i]\n");
    }
}
```

**This reads?** \_\_\_\_\_

**Will Match?** \_\_\_\_\_

\_\_\_\_\_

\_\_\_\_\_

# Regular Expressions

---

## Quantifiers

A subtle lesson (one you may not fully appreciate until you begin writing regular expressions)

### Will they match?

```
my $str="flavour";
```

```
my $str="flavor";
```

```
$str =~ /flavou?r/)
```

Yes

Yes

```
$str =~ /flavou*r/)
```

```
$str =~ /flavou+r/)
```

# Regular Expressions

---

## Character classes

`Gre[ea]t`

**This reads**, match “G” followed by “r”, followed by “e”, followed by either an “e” or an “a”, followed by a “t”. This expression will match both “Greet” and “Great”.

Of utmost Importance: **A character class will only match a single character!!!**  
For example, in our example above it is either an “e” or an “a”, NOT an “e” followed by an “a”.

Very Useful Note: You can also use \*, ?, + with a character class.

## Character classes

### Being lazy

`[0-9]` means the same as `[0123456789]`

`[A-D]` means the same as `[ABCD]`

### Parenthesis and Alternation

`(T|True|Y|Yes)`

**This reads**, match “T” or “True” or “Y” or “Yes”.

## In class exercise

---

- Operand1=Operand2;
- Operand1 =Operand2;
- Operand1 = Operand2;
- Operand1 =           Operand2;
- Operand1 = Operand2;

```
/Operand[0-9]* *= *Operand[0-9]*;/
```

**Question:** Will this statement match the five items above? \_\_\_\_\_

Break into groups of 3 and figure this out. You must be able to read the regular expression and tell us why it will or will not match the 5 lines.

## In class exercise

```
my @arr=(  
    "Operand1=Operand2;"      ,  
    "Operand1 =Operand2;"    ,  
    "Operand1 = Operand2;"   ,  
    "Operand1 =      Operand2;" ,  
    "Operand1 = Operand2;"   ,  
    "Operand211 = Operand22;" ,  
    "Operand1 = Operand2  ;"  ,  
    "  Operand1 = Operand2  ;"  
);
```

**Rewrite regex2.pl so that matches the last two items in the array.**

## In class exercise

---

11/28/47

11-28-47

`[0-9]* (\/|-) [0-9]* (\/|-) [0-9]*`

**This reads?** \_\_\_\_\_

**Will it match the two dates?** \_\_\_\_\_

## In class exercise

---

```
11/28/47
```

```
11-28-47
```

```
1/2/01
```

```
-12-98
```

```
11//54
```

```
//
```

```
--
```

**Rewrite regex3.pl so that it only matches valid dates.**

### The . operator

Will match any character (except newline)

# Regular Expressions

---

## Anchors away

<b>^</b>	Matches at the beginning of the line.
<b>\$</b>	Matches at the end of the line.
<b>\b</b>	Match a word boundary.
<b>\B</b>	Match a non-word boundary.
<b>\A</b>	Matches only at the beginning of a string.
<b>\Z</b>	Match only at the end of a string, or before a newline at the end.
<b>\z</b>	Match only at the end of a string.

## Regular Expressions

---

### Anchors away

“\w” represents the character anchor, and “\W” represents the non character anchor.

A	=~	/\w\
a	=~	/\w\
8	=~	/\w\
_	=~	/\w\
+	=~	/\W\
-	=~	/\W\
@	=~	/\W\

These all match

## In class exercise

---

"This was on." =~ /is/

**Match?** \_\_\_\_\_

"This was on." =~ /\bis/

**Match?** \_\_\_\_\_

"This was on." =~ /is\b/

**Match?** \_\_\_\_\_

"This was on." =~ /\bis\b/

**Match?** \_\_\_\_\_

## In class exercise

---

# Negation

Regex5\_1.pl

# Regular Expressions

---

## Case Insensitive

```
/<some reg ex goes here>/i
```

Just add an “i” after your regex to make it case insensitive. Keep in mind this adds a lot more work for the engine. Only use this where you truly need to make you regex case insensitive, or where it is impractical to localize case.

```
/hello/i
```

Will match

```
Hello
```

```
hello
```

```
HELLO
```

```
Hello
```

Manual: p72

## Assignments

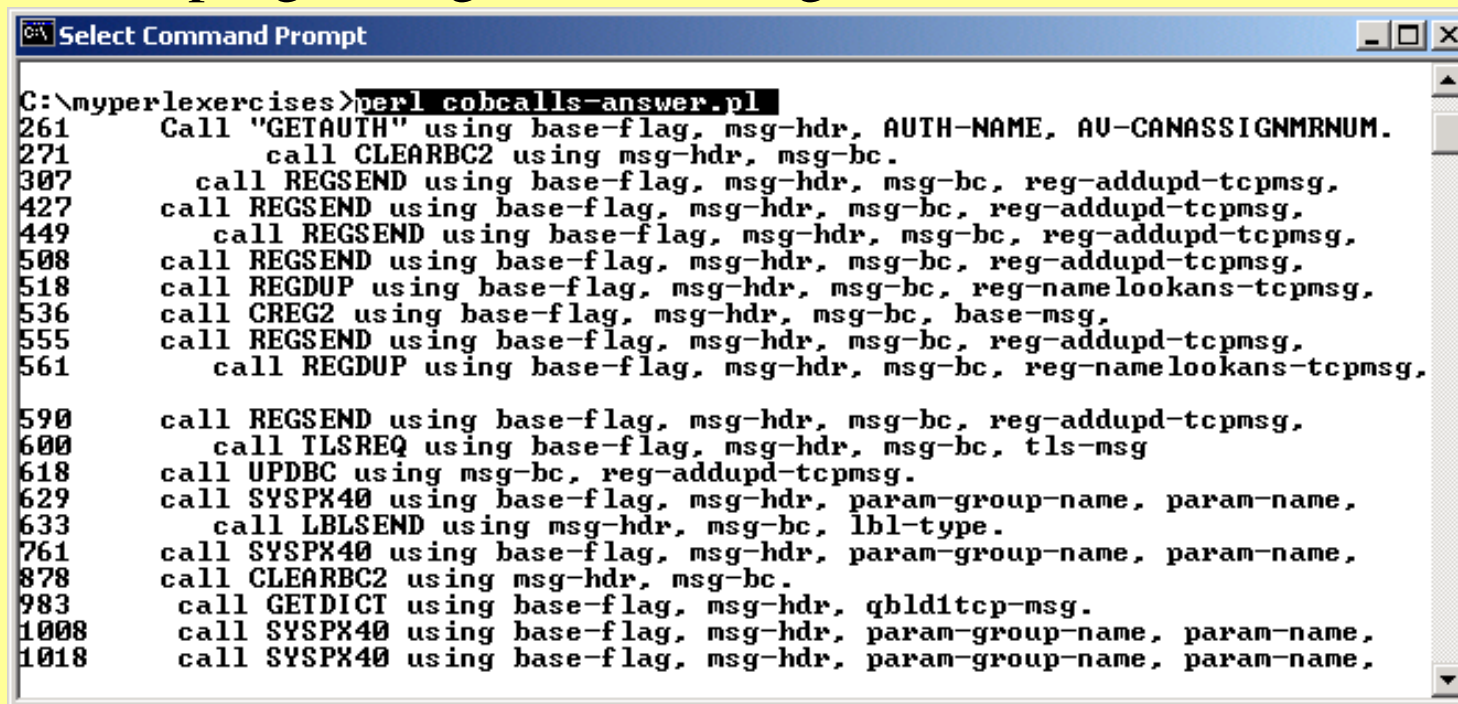
---

The following are a series of assignments for you to code. Pick **one** of them to do. You certainly can do more than one, and the added work can only serve to aid in your proficiency with Perl.

## Assignments

### Return calls from a Cobol program

- Do not return any comment lines
- Do not return calls to version control
- Do not return word call from working storage
- Test program against file “reg.txt”



```
Select Command Prompt
C:\myperl\exercis>perl cobcalls-answer.pl
261 Call "GETAUTH" using base-flag, msg-hdr, AUTH-NAME, AU-CANASSIGNMRNUM.
271 call CLEARBC2 using msg-hdr, msg-bc.
307 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
427 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
449 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
508 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
518 call REGDUP using base-flag, msg-hdr, msg-bc, reg-namelookans-tcpmsg,
536 call CREG2 using base-flag, msg-hdr, msg-bc, base-msg,
555 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
561 call REGDUP using base-flag, msg-hdr, msg-bc, reg-namelookans-tcpmsg.

590 call REGSEND using base-flag, msg-hdr, msg-bc, reg-addupd-tcpmsg,
600 call TLSREQ using base-flag, msg-hdr, msg-bc, tls-msg
618 call UPDBC using msg-bc, reg-addupd-tcpmsg.
629 call SYSPX40 using base-flag, msg-hdr, param-group-name, param-name,
633 call LBLSEND using msg-hdr, msg-bc, lbl-type.
761 call SYSPX40 using base-flag, msg-hdr, param-group-name, param-name,
878 call CLEARBC2 using msg-hdr, msg-bc.
983 call GETDICT using base-flag, msg-hdr, qblditcp-msg.
1008 call SYSPX40 using base-flag, msg-hdr, param-group-name, param-name,
1018 call SYSPX40 using base-flag, msg-hdr, param-group-name, param-name,
```

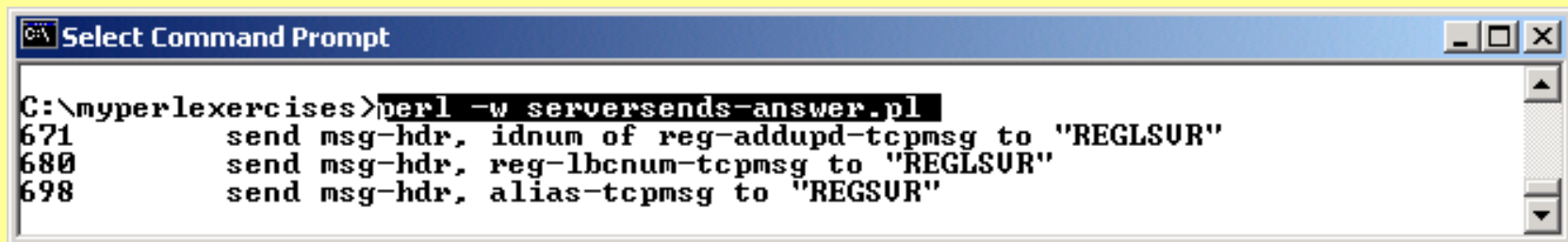
cobcalls-ANSWER.pl

## Assignments

---

### Return server sends from a Cobol program

- Do not return any comment lines
- Do not return send from working storage
- Test program against file “reg.txt”



```
CA Select Command Prompt
C:\myperlexercises>perl -w serversends-answer.pl
671      send msg-hdr, idnum of reg-addupd-tcpmsg to "REGLSUR"
680      send msg-hdr, reg-lbcnum-tcpmsg to "REGLSUR"
698      send msg-hdr, alias-tcpmsg to "REGSUR"
```

serversends-ANSWER.pl

## Assignments

---

### Extract a named section from a file and dump into another file

- Write a program that extracts a ?section from a file and dumps it into another file.
- Test the program against the file ddl.txt
- Make sure to test that program works the same when extracting the last ?section in a file as it does for extracting all other ?section's.

**This is the most challenging of the exercises. I would encourage you to try it as it will really test how adept you are becoming with Perl.**

sect.pl