CSCI3180 Principle of Programming Languages

Tutorial 7: Introduction to Perl

TA Information

- LIN Huaijia
  - Office:
    - SHB 1026
  - Office Hours:
    - Tuesday 2:30pm – 3:30pm
  - linhj@cse.cuhk.edu.hk

Outline

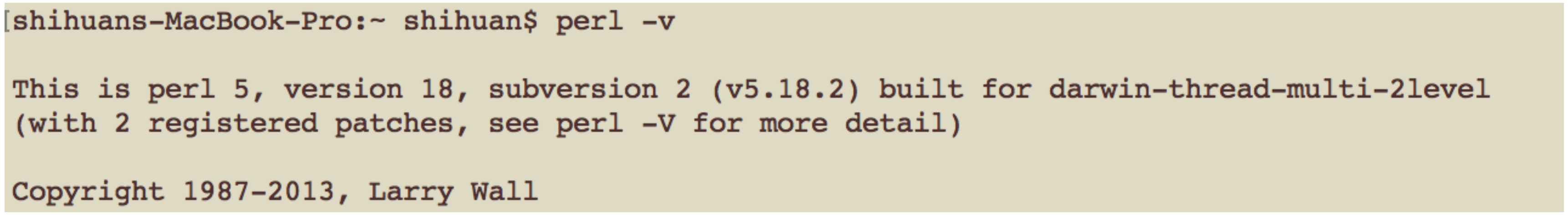
- Programming environment
- Basic properties of Perl
- Object oriented programming in Perl

Perl

- Perl programming language
  - is an interpreted language
  - is dynamic, general-purpose, and object oriented
- Special features
  - is a powerful scripting language
    - borrow features from many languages
      - the C, shell script(sh), etc.
  - supports both static and dynamic scoping

Perl Installation

- Whether have already installed? Which version?
  - perl –v

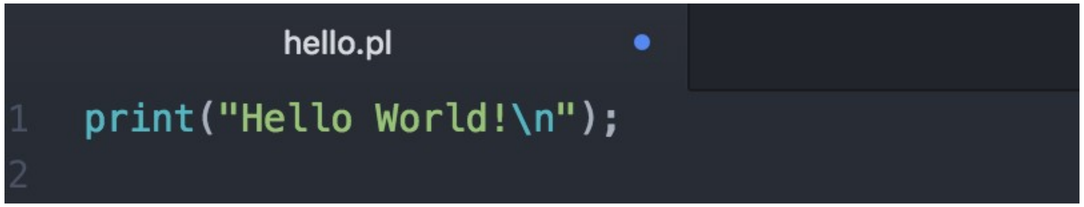


```
[shihuans-MacBook-Pro:~ shihuan$ perl -v

This is perl 5, version 18, subversion 2 (v5.18.2) built for darwin-thread-multi-2level
(with 2 registered patches, see perl -V for more detail)

Copyright 1987-2013, Larry Wall
```

- Download website
  - https://www.perl.org/get.html
- Perl v5.18.2 for assignment 3
  - Perl 5, version 18, subversion 2

Running Your First Perl Program

- Open any text editor, e.g. Notepad++, Sublime Text
- Type in the following code

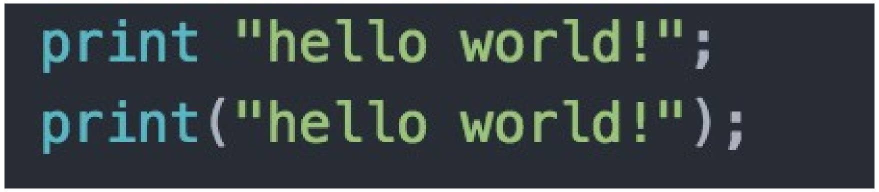


```perl
print("Hello World!\n");
```

- Save it as “hello.pl” in your home folder

Call function with or without ()

- Call function with or without ()

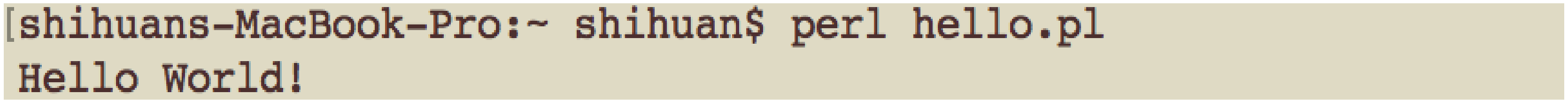


```
print "hello world!";
print("hello world!");
```

- Choose the way you like

Running Your First Perl Program

- Windows users: Open the Command Prompt
- Mac users: Open the Terminal
- Type “perl hello.pl”

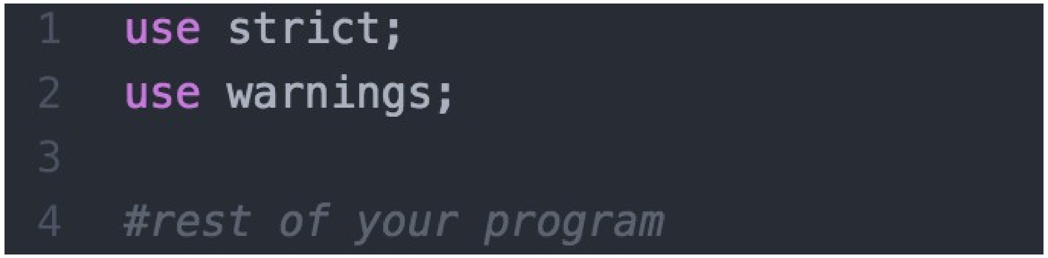


```
[shihuans-MacBook-Pro:~ shihuan$ perl hello.pl
 Hello World!
```

- If you use Sublime Text, you can build it with build system

Use Strict and Use Warnings

- Compiler flags that instruct Perl to behave in a stricter way
- Help you avoid a number of common programming mistakes



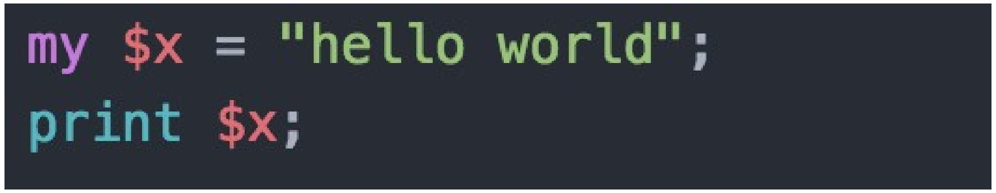
```perl
use strict;
use warnings;

#rest of your program
```

Perl Variables

- Scalar
  - preceded by a dollar sign $
  - store either a string, a number or a reference
- Array
  - preceded by sign @
  - store ordered lists of scalars
- Hash
  - preceded by sign %
  - store sets of key/value pairs

Scalar

- Preceded by a dollar sign $
- Store either a string, a number or a reference



```
my $x = "hello world";
print $x;
```

“my” in Perl

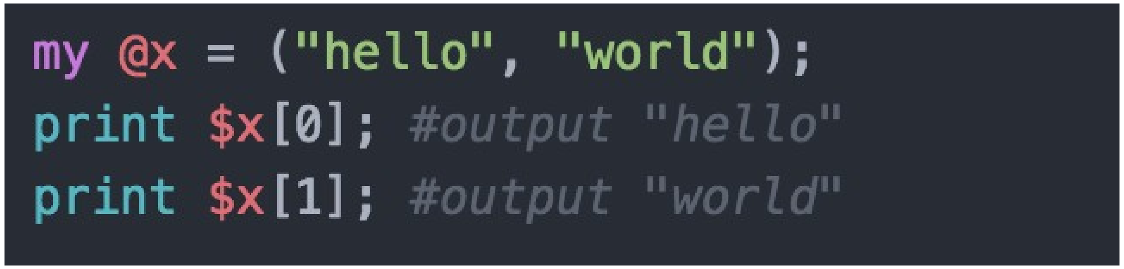
A *my* declares the listed variables to be local (lexically) to the enclosing block, file

Next tutorial..

- Statically scoped variables
  - Declared with keyword “my”
- Statically scoped global variables
  - Declared with keyword “our”
- Dynamically scoped variables
  - Declared with keyword “local”

Array

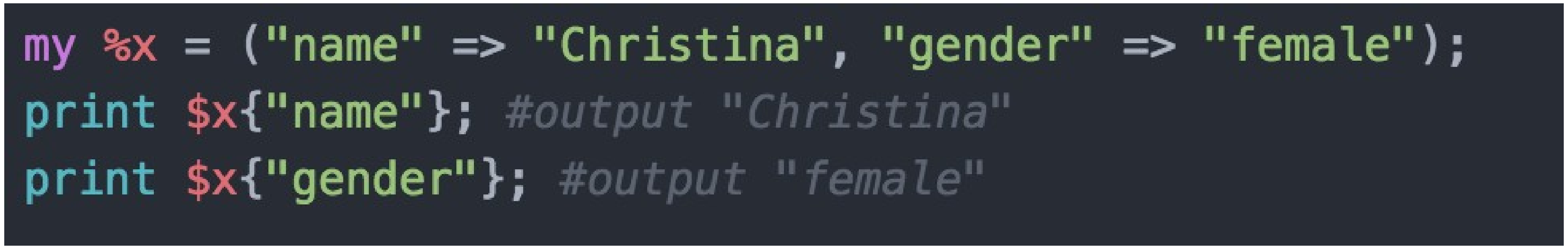
- Preceded by sign @
- Store lists of scalars (strings, numbers or references)



```perl
my @x = ("hello", "world");
print $x[0]; #output "hello"
print $x[1]; #output "world"
```

Hash

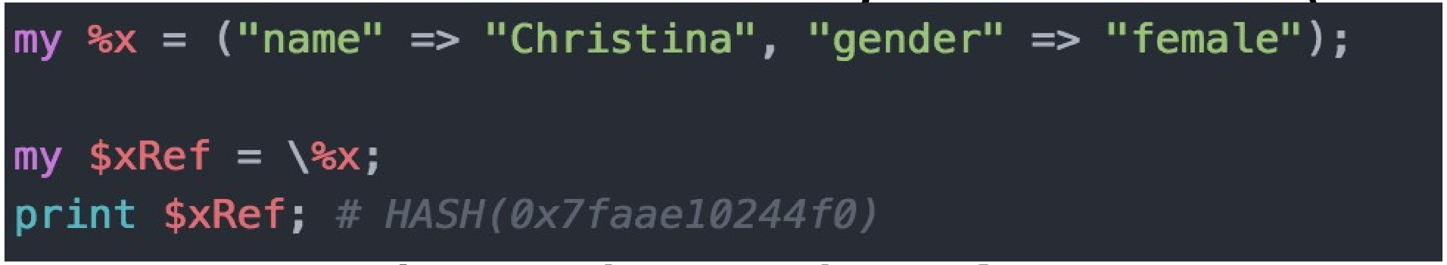
- Preceded by sign %
- Store sets of key/value pairs



```perl
my %x = ("name" => "Christina", "gender" => "female");
print $x{"name"}; #output "Christina"
print $x{"gender"}; #output "female"
```

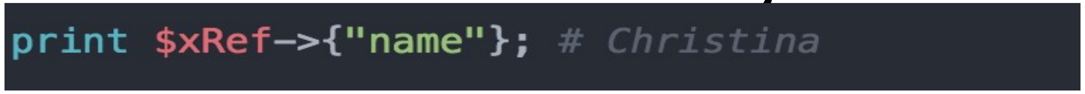
References

- Create reference using backslash \



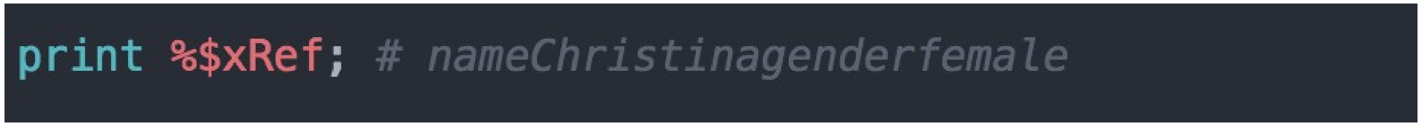
```perl
my %x = ("name" => "Christina", "gender" => "female");

my $xRef = \%x;
print $xRef; # HASH(0x7faae10244f0)
```

- Access values through reference



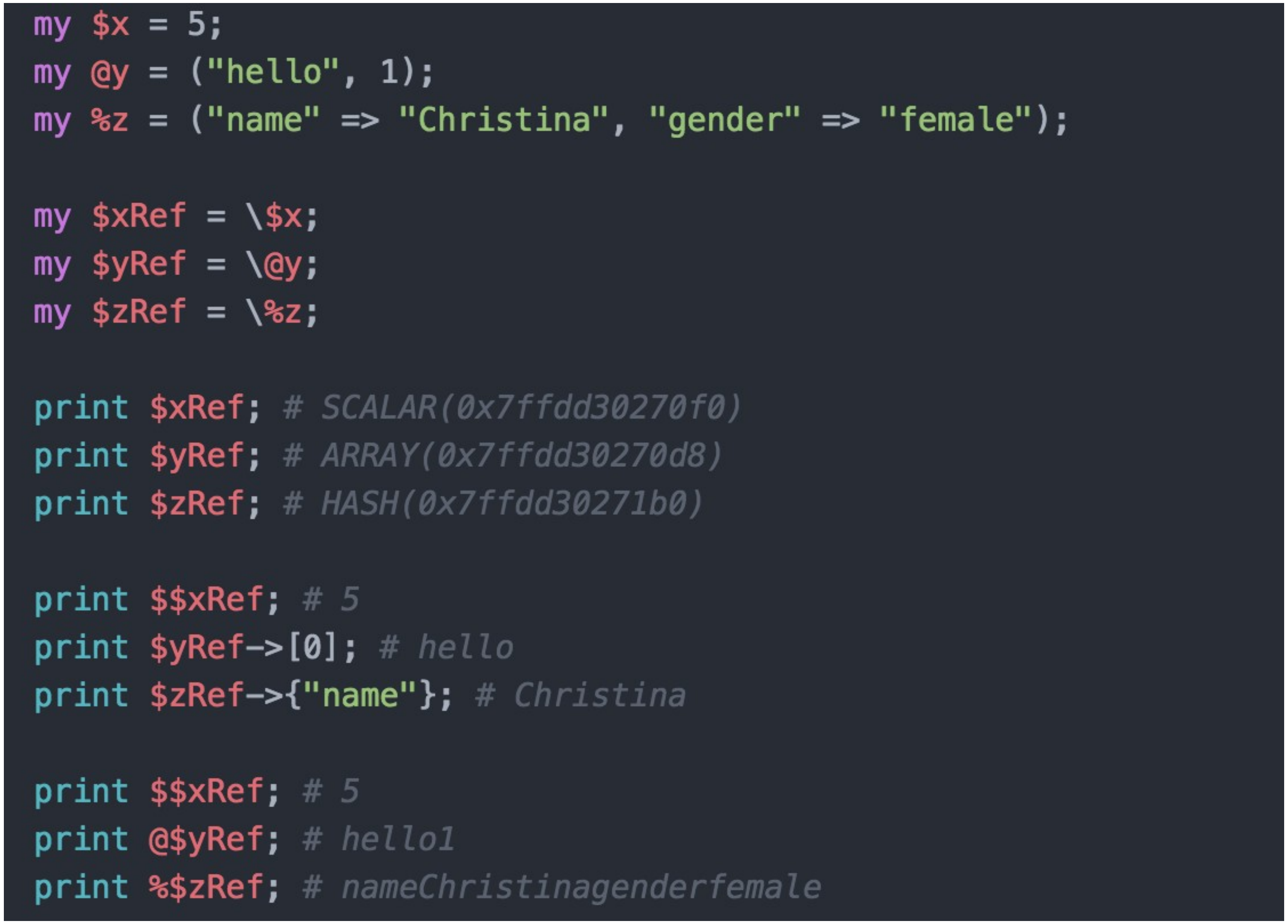
```perl
print $xRef->{"name"}; # Christina
```

- Dereference a reference using $, @ or % as prefix



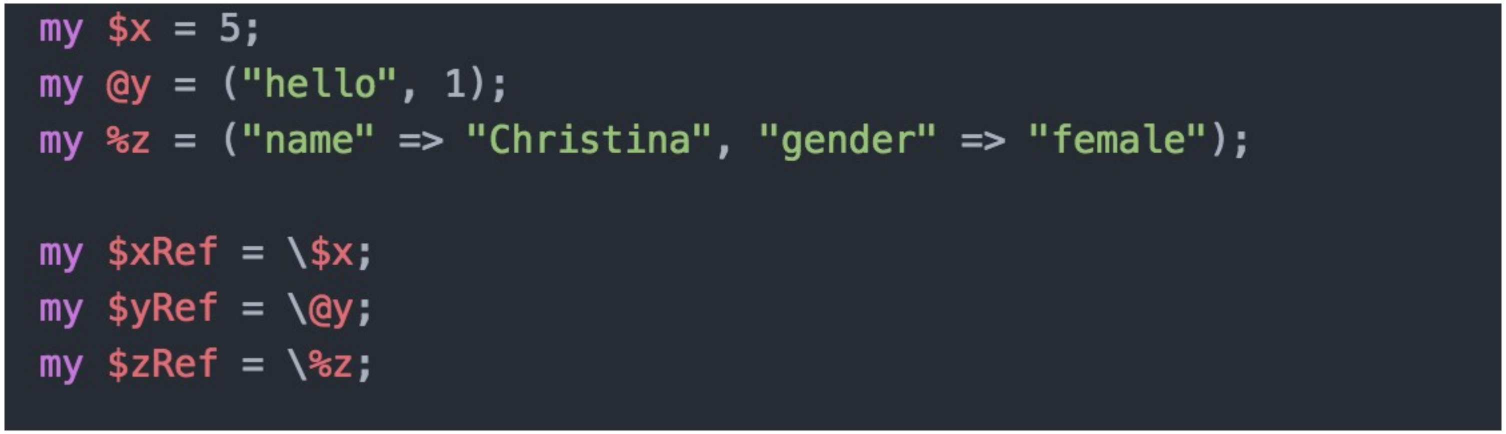
```perl
print %$xRef; # nameChristinagenderfemale
```

References

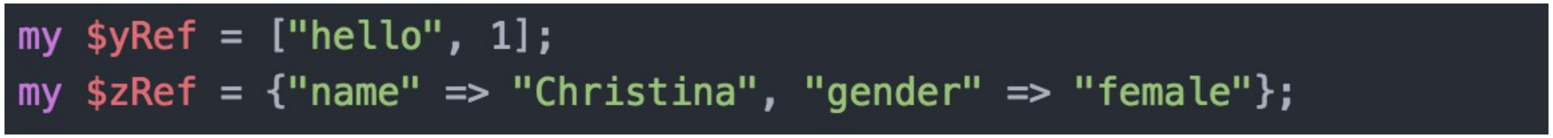


```perl
my $x = 5;
my @y = ("hello", 1);
my %z = ("name" => "Christina", "gender" => "female");

my $xRef = \$x;
my $yRef = \@y;
my $zRef = \%z;

print $xRef; # SCALAR(0x7ffdd30270f0)
print $yRef; # ARRAY(0x7ffdd30270d8)
print $zRef; # HASH(0x7ffdd30271b0)

print $$xRef; # 5
print $yRef->[0]; # hello
print $zRef->{"name"}; # Christina

print $$xRef; # 5
print @$yRef; # hello1
print %$zRef; # nameChristinagenderfemale
```

References



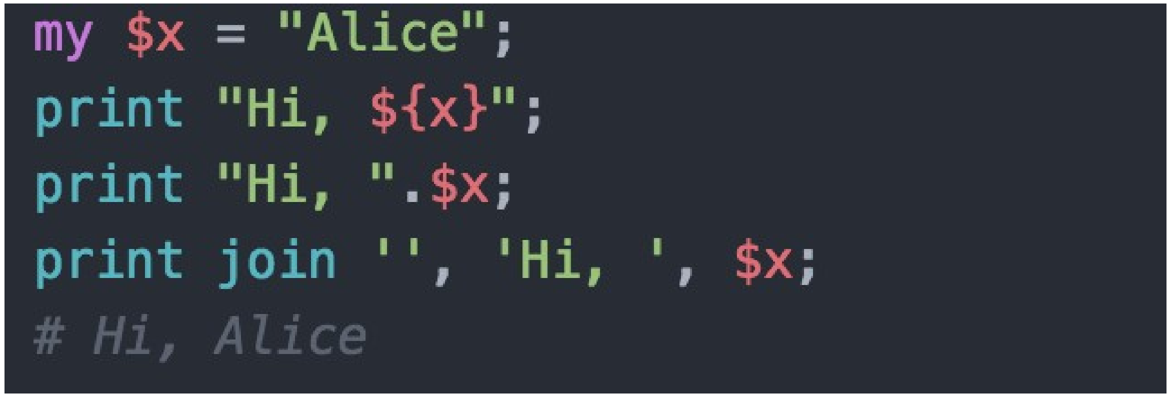
```perl
my $x = 5;
my @y = ("hello", 1);
my %z = ("name" => "Christina", "gender" => "female");

my $xRef = \$x;
my $yRef = \@y;
my $zRef = \%z;
```



```perl
my $yRef = ["hello", 1];
my $zRef = {"name" => "Christina", "gender" => "female"};
```

String Concatenation

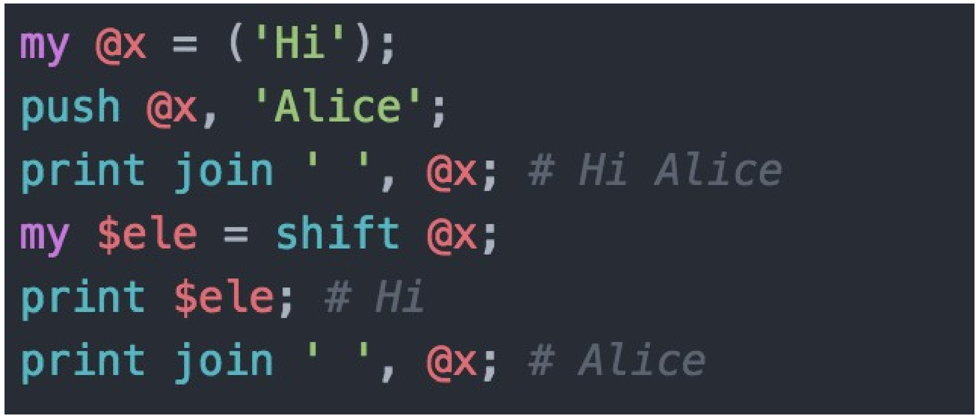
- Using ${name}
- Using dot operator
- Using join function



```
my $x = "Alice";
print "Hi, ${x}";
print "Hi, ".$x;
print join '', 'Hi, ', $x;
# Hi, Alice
```

Some useful functions for Array

- Push
  - add an element to the back of the array
- Shift
  - pop and return the first element in the array

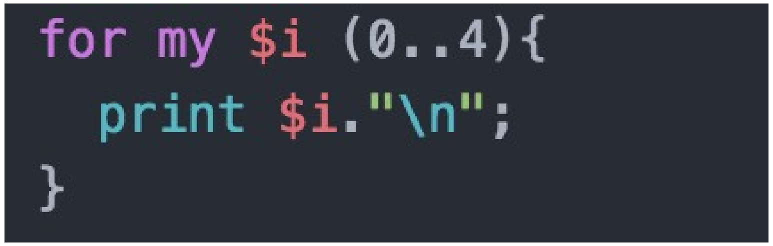


```perl
my @x = ('Hi');
push @x, 'Alice';
print join ' ', @x; # Hi Alice
my $ele = shift @x;
print $ele; # Hi
print join ' ', @x; # Alice
```

For Loop

- Traditional numerical for loop in Perl
  - For my $i (0..n)
- e.g., print the numbers from 0 through 4

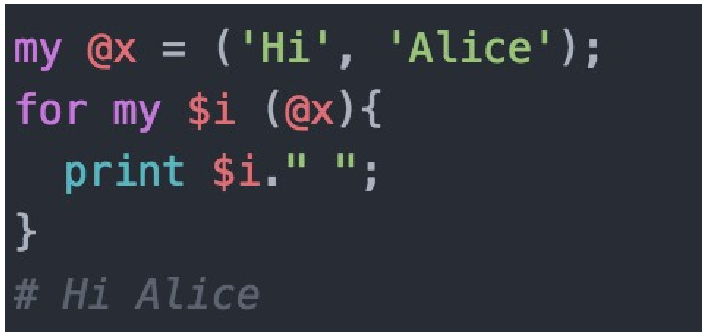
Perl code:



```
for my $i (0..4){
  print $i."\n";
}
```

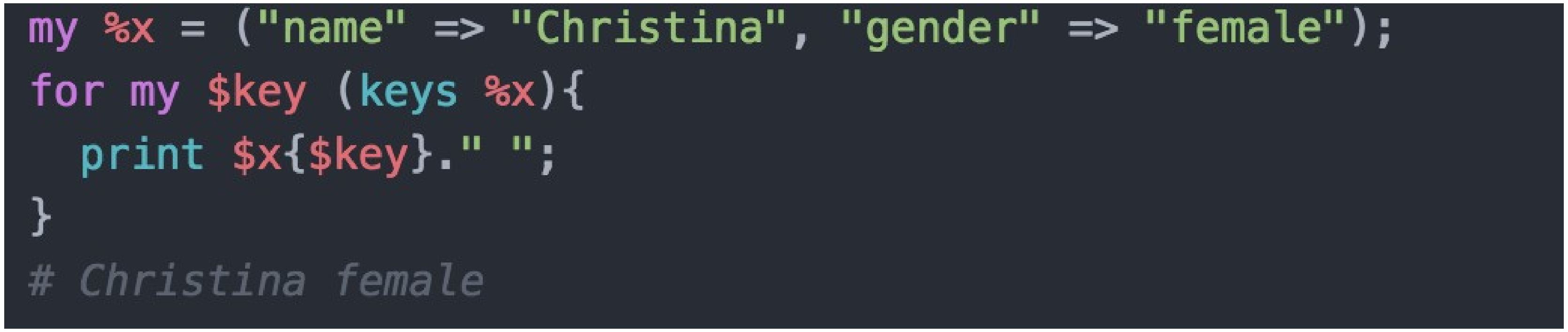
For Loop

- Loop through an array



```
my @x = ('Hi', 'Alice');
for my $i (@x){
  print $i." ";
}
# Hi Alice
```

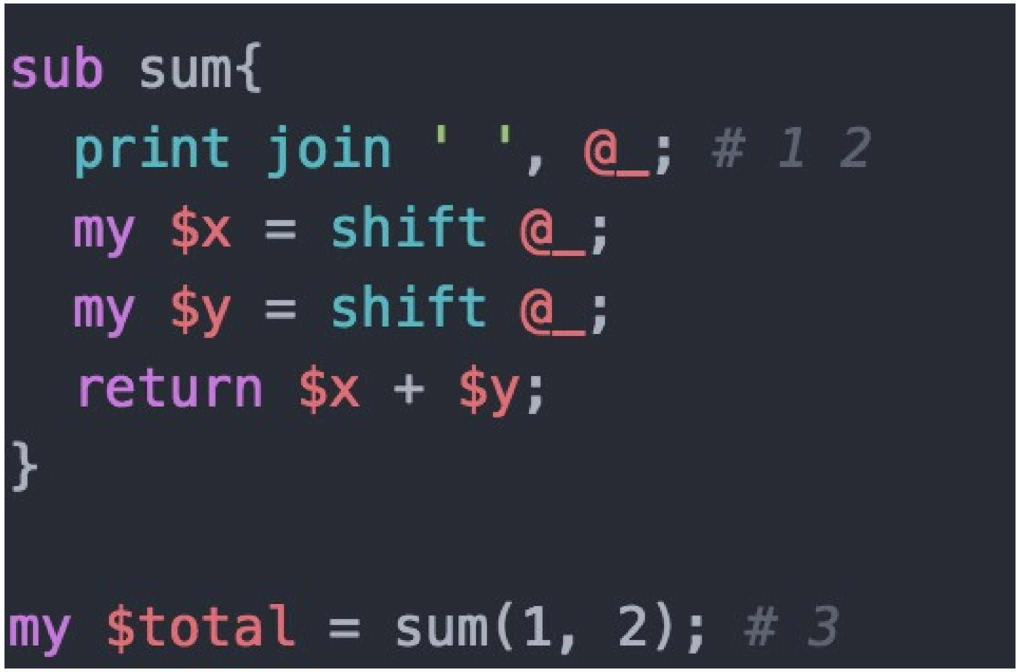
- Loop through an hash



```
my %x = ("name" => "Christina", "gender" => "female");
for my $key (keys %x){
  print $x{$key}." ";
}
# Christina female
```

Functions or Subroutines

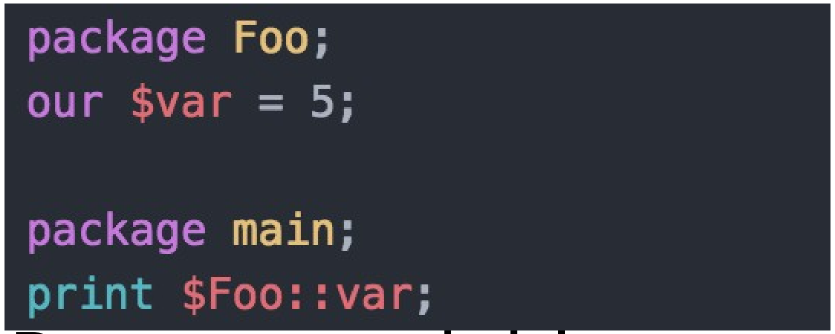
- Declare a subroutine with **sub**
- Retrieve parameters in **@_**



```
sub sum{
  print join ' ', @_; # 1 2
  my $x = shift @_;
  my $y = shift @_;
  return $x + $y;
}

my $total = sum(1, 2); # 3
```

Package in Perl

- A collection of code which lives in its own namespace
- Explicitly refer to functions and variables in a package using “::”

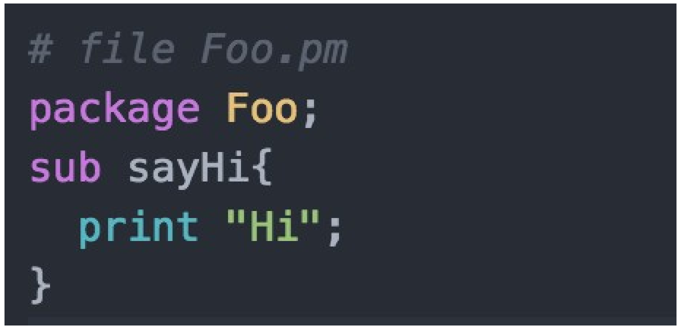


```perl
package Foo;
our $var = 5;

package main;
print $Foo::var;
```

- Prevent variable name collisions between packages

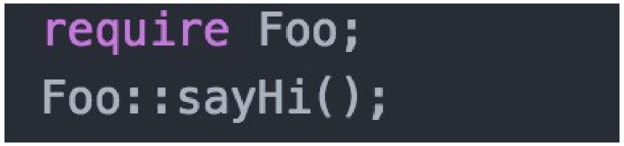
Perl Modules

- A Good Practice to use Perl module
  - one single package in each Perl module
  - a .pm as extension, where a is the name of the package



```perl
# file Foo.pm
package Foo;
sub sayHi{
  print "Hi";
}
```

- Use **require** and **use** to load a module.



```perl
require Foo;
Foo::sayHi();
```

Object Oriented Programming

- Object
  - a reference to a data type that knows what class it belongs to
  - **bless** a reference as a object of a certain class
- Class
  - a package that contains the corresponding methods required to create and manipulate objects
- Method
  - a subroutine, defined within the package
  - the first argument to the method is an object reference or a package name

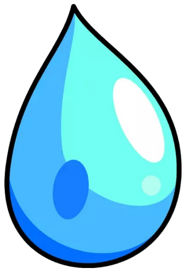
Bless

- Bless a reference as a object of a certain class

```
# “Pokemon.pm”
package Pokemon;
sub sayHi{
    print “Hi”;
}
```

```
use Pokemon;

my $x = {“hp” => 80};
my $obj = bless( $x, “Pokemon”);
$obj->sayHi(); # Hi
```

Define a class



```
package WaterTypePokemon;

sub new{
    my $class = shift @_;
    my $hp = shift @_;
    my $weight = shift @_;

    my $object = bless {"HP" => $hp,
        "weight" => $weight}, $class;
    return $object;
}
```

Get arguments

Define a method

Create an object

```
$pkm1 = WaterTypePokemon->new(80, 30);
```

Define a class

```
package WaterTypePokemon;

sub new{
    my $class = shift @_;
    my $hp = shift @_;
    my $weight = shift @_;

    my $object = bless {"HP" => $hp,
        "weight" => $weight}, $class;
    return $object;
}
```

Get arguments

Define a method

Create an object

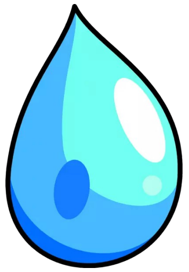
```
$pkm1 = WaterTypePokemon->new(80, 30);
```

Define a class

```
package WaterTypePokemon;

sub new{
    .....
    my $object = bless {”HP” => $hp,
        ”weight” => $weight}, $class;
    return $object;
}

sub exercise{
    my $self = shift @_;
    $self->{“weight”} -= 5;
}
```

Define another method

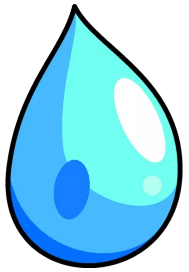
Access an objects’ attribute



```
package WaterTypePokemon;
sub new{
    .....
    return $object;
}
sub exercise{
    my $self = shift @_;
    $self->{“weight”} -= 5;
}

$pkm1 = WaterTypePokemon->new(80, 30);
$pkm1->{”HP”} = 90;
```

Access an object’s attribute

Call an objects’ method



```
package WaterTypePokemon;
sub new{
    .....
    return $object;
}
sub exercise{
    my $self = shift @_;
    $self->{“weight”} -= 5;
}

$pkm1 = WaterTypePokemon-
>new(80, 30);
$pkm1->{”HP”} = 90;
$pkm1->exercise();
```

Call an object’s method
Perl will add the object itself (pkm1) as the first argument to call exercise.

Inheritance

```
package Psyduck;
use parent('WaterTypePokemon'):
```

Psyduck extends WaterTypePokemon class
Inherit methods of its super class

Inheritance

```
package Psyduck;
use parent('WaterTypePokemon'):

Sub new{
    my $class = shift @_;
    my $hp = shift @_;
    my $weight = shift @_;
    my $haircount = shift @_;

    my $object = bless {"HP" => $hp,
        "weight" => $weight,
        "HairCount" => $haircount},$class;
    return $object;
}
```

Override its superclass's function

Inheritance

```
package Psyduck;
use parent('WaterTypePokemon'):

Sub new{
    .....
    return $object;
}
Sub scratch{
    my $self = shift @_;
    my $target = shift @_;

    $target->{"HP"} -= 5;
}
```

Define another method

Learning Resources

- Perl tutorial website
- https://www.tutorialspoint.com/perl/index.htm

- Install Perl modules
  - http://www.cpan.org/modules/index.html

END

- Q&A