Tutorial 7: Dynamic & Static Scoping

Yanjing Ren & Huancheng Puyang

yjren22@cse.cuhk.edu.hk

hcpuyang22@cse.cuhk.edu.hk

Outline

- An overview of variables in Perl
- Lexical variables
  - Prefix: my
- Package variables
  - Prefixes: local, our

Static and Dynamic Scoping in Perl

- Lexical variable – static scoping
  - Declared with keyword “my”
- Package variables – both static and dynamic scoping
  - Statically scoped package variables
    - Declared with keyword “our”
  - Dynamically scoped package variables
    - Declared with keyword “local”

Lexical Variables

- Declared with keyword my
  - Example(s): `my $var = 1.8;`
- Scoping type - static scoping
  - The scope of a lexical variable is determined at compile time
  - It cannot be accessed from anywhere outside its enclosing scope

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

```
{
    my $x= "hello";
}
print $x;
```

Lexical Variables

- Declared with keyword my
  - Example(s):
```
my $var = 1.8;
```
- Scoping type - static scoping
  - The scope of a lexical variable is determined at compile time
  - It cannot be accessed from anywhere outside its enclosing scope

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

```
{
    my $x= "hello";
}
print $x;
```
<-No output

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined

```
sub s1{
  my $x = "x is defined in s1";
  sub s2{
    s3();
    sub s3{
      print $x; # x is defined in s1
    }
  }
  s2();
}

s1();
```

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., search in s3 first

```
sub s1{
  my $x = "x is defined in s1";
  sub s2{
    s3();
    sub s3{
      print $x; # x is defined in s1
    }
  }
  s2();
}

s1();
```

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., then search in s2

```
sub s1{
  my $x = "x is defined in s1";
  sub s2{
    s3();
    sub s3{
      print $x; # x is defined in s1
    }
  }
  s2();
}

s1();
```

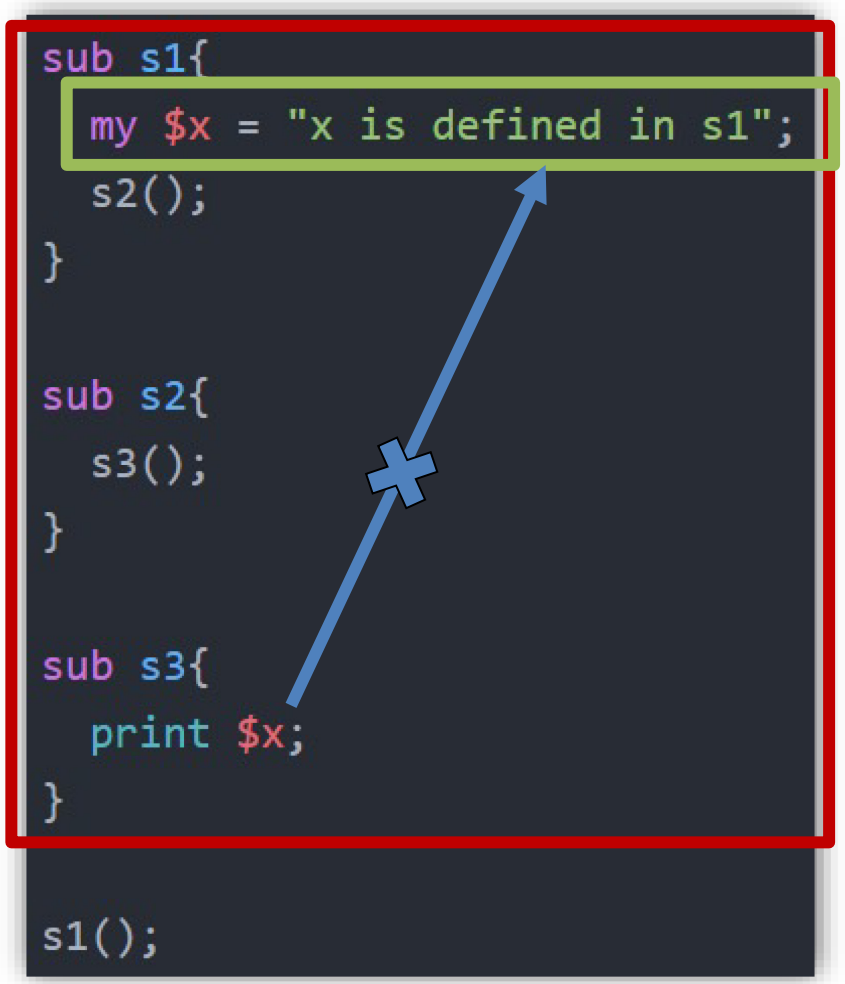
Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., Finally search in s1
  - Output: “x is defined in s1”

```
sub s1{
  my $x = "x is defined in s1";
  sub s2{
    s3();
    sub s3{
      print $x; # x is defined in s1
    }
  }
  s2();
}

s1();
```

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., non-nested three functions
  - Output: nothing



Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., two scalar-> x and y
  - Call chain: aSub()-> bSub()
  - Output 1: “hello”

```
my $x= "hello";

sub aSub {
    my $y= "world ";
    bSub();
    print $x; # hello
}

sub bSub {
    print $x; # hello
    print $y; #
    my $x = "good";
    print $x; # good
}

aSub();
```

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., two scalar-> x and y
  - Call chain: aSub()-> bSub()
  - Output 1: “hello”
  - Output 2: nothing

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., two scalar-> x and y
  - Call chain: aSub()-> bSub()
  - Output 1: “hello”
  - Output 2: nothing
  - Output 3: “good” (scalar changed)

```
my $x= "hello";

sub aSub {
    my $y= "world ";
    bSub();
    print $x; # hello
}

sub bSub {
    print $x; # hello
    print $y; #
    my $x = "good";
    print $x; # good
}

aSub();
```

Lexical Variables

- Rules for static scoping
  1. Search in the local function (the function which is running now)
  2. Search in the function (or scope) in which that function was defined
- E.g., two scalar-> x and y
  - Call chain: aSub()-> bSub()
  - Output 1: “hello”
  - Output 2: nothing
  - Output 3: “good” (scalar changed in bSub())
  - Output 4: “hello”

```
my $x= "hello";

sub aSub {
    my $y= "world ";
    bSub();
    print $x; # hello
}

sub bSub {
    print $x; # hello
    print $y; #
    my $x = "good";
    print $x; # good
}

aSub();
```

Package Variables

- Package
  - A collection of code which lives in its own namespace
- Each package maintains a symbol table
  - Symbol Table: a hash table
    - Whenever a new variable or subroutine is defined, a new entry is added to the table for that package
- Why we have package variables?
  - Prevent variable name collisions between packages
  - Explicitly refer to functions and variables in a package using “::”

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

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

Package Variables

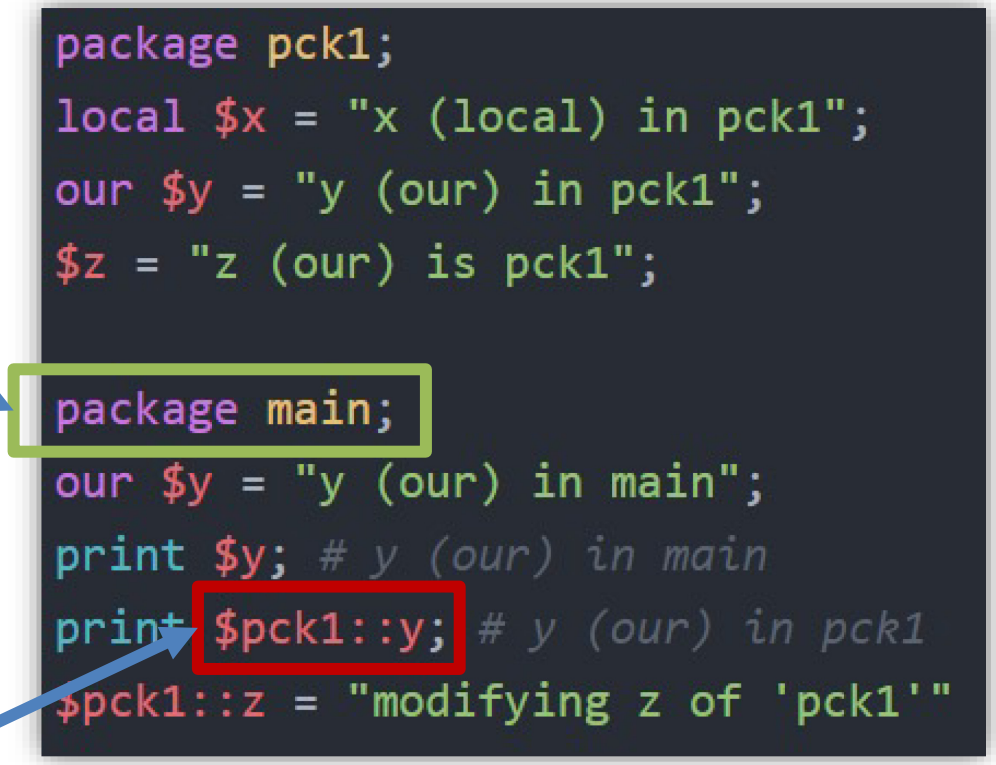
- Variables without any prefix is a package variable (our)
- Package variables have the scope of package in which they were declared
- A package variable can be accessed or modified from other packages by specifying the full package name

```
package pck1;
local $x = "x (local) in pck1";
our $y = "y (our) in pck1";
$z = "z (our) is pck1";

package main;
our $y = "y (our) in main";
print $y; # y (our) in main
print $pck1::y; # y (our) in pck1
$pck1::z = "modifying z of 'pck1'"
```

Package Variables

- All subroutines and package variables must have a package.
- The root namespace is main
- Declared with keyword our
  - package variables, and thus automatically
    - Global variables
    - Public: can be accessed by outside
    - They can be accessed outside the package (or lexical scope) with the qualified namespace
    - $package_name::variable

Static Scoping



```
package pck1;
local $x = "x (local) in pck1";
our $y = "y (our) in pck1";
$z = "z (our) is pck1";

package main;
our $y = "y (our) in main";
print $y; # y (our) in main
print $pck1::y; # y (our) in pck1
$pck1::z = "modifying z of 'pck1'"
```

Package Variables

- Declared with keyword local
  - Variables prefixed with local are package variables with temporality property
  - A variable declared with local hides an existing variable by masking it with a temporary value as long as its lexical scope exists
    - **Different from other package variables**
  - Variables declared with local are visible in called subroutines (recall the calling sequences)
    - **Different from lexical variables**

Dynamic Scoping

Package Variables

- How to search for a variable with a given name in dynamic scoping?
  - Search in the local function first
  - Then search in the function that called the local function
  - Then search in the function that called that function
  - And so on, up the call stack

Package Variables: Comparison

- Rules for static scoping
  - Search in the local function (the function which is running now)
  - Search in the function (or scope) in which that function was defined
  - Search in the function (or scope) in which that function was defined…
  - So forth

Static Scoping

- Rules for dynamic scoping
  - Search in the local function
  - Search in the function that called the local function
  - Search in the function that called that function…
  - and so on, up the call stack.

Dynamic Scoping

Package Variables: Example

- **An example with both static & dynamic scoping**

```
package Foo;

our $x = "hello ";
our $y = "world ";

sub sub1{
    print join '', $x, $y;
}
```

```
require "./Foo.pm";

sub sub2{
    local $Foo::x = "hi ";
    $Foo::y = "you ";
    Foo::sub1();
}

Foo::sub1();
sub2();
Foo::sub1();
```

Package Variables: Example

- **An example with both static & dynamic scoping**

```
package Foo;

our $x = "hello ";
our $y = "world ";

sub sub1{
    print join '', $x, $y;
}
```

```
require "./Foo.pm";

sub sub2{
    local $Foo::x = "hi ";
    $Foo::y = "you ";
    Foo::sub1();
}

Foo::sub1();
sub2();
Foo::sub1();
```

=> hello world

Package Variables: Example

- **An example with both static & dynamic scoping**

```
package Foo;

our $x = "hello ";
our $y = "world ";

sub sub1{
    print join '', $x, $y;
}
```

```
require "./Foo.pm";

sub sub2{
    local $Foo::x = "hi ";
    $Foo::y = "you ";
    Foo::sub1();
}

Foo::sub1();
sub2();
Foo::sub1();
```

X-> hi

Package Variables: Example

- **An example with both static & dynamic scoping**

```
package Foo;

our $x = "hello ";
our $y = "world ";

sub sub1{
    print join '', $x, $y;
}
```

```
require "./Foo.pm";

sub sub2{
    local $Foo::x = "hi ";
    $Foo::y = "you ";
    Foo::sub1();
}

Foo::sub1();
sub2();
Foo::sub1();
```

X-> hi (local)

Y-> you (global)

=> hi you

Package Variables: Example

- **An example with both static & dynamic scoping**

```
package Foo;

our $x = "hello ";
our $y = "world ";

sub sub1{
    print join '', $x, $y;
}
```

```
require "./Foo.pm";

sub sub2{
    local $Foo::x = "hi ";
    $Foo::y = "you ";
    Foo::sub1();
}

Foo::sub1();
sub2();
Foo::sub1();
```

X-> hi (local)

Y-> you (global)

=> hello you

Package Variables: Example

- An example with both static & dynamic scoping + Lexical variables

```
sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "hello ";
    my $y = "world ";
    sub1();
}

sub1();
sub2();
sub1();
```

Dynamic scoping

Lexical variable

Package Variables: Example

- **An example with both static & dynamic scoping + Lexical variables**
  - No output since $x and $y are not defined

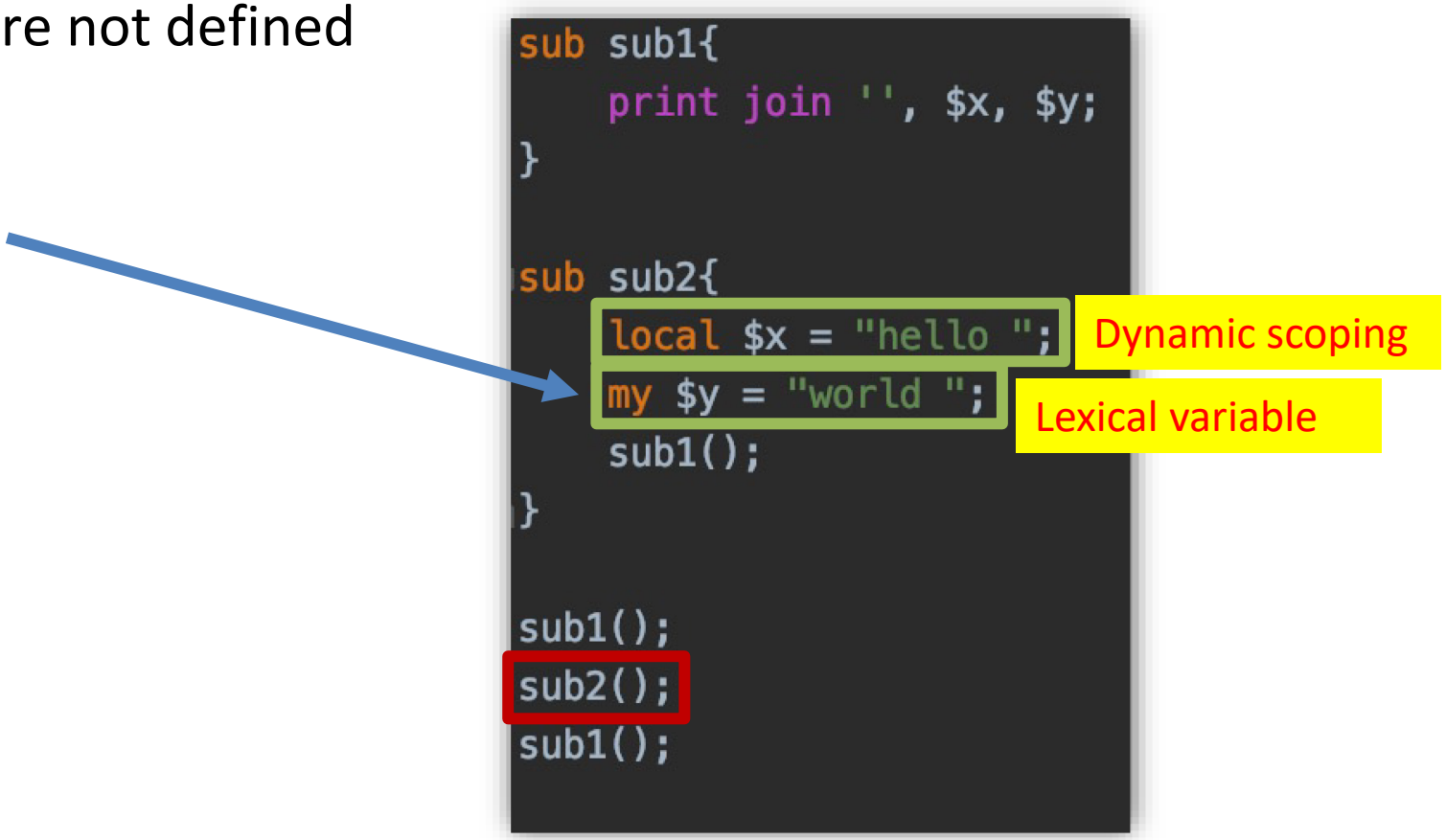
```
sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "hello ";
    my $y = "world ";
    sub1();
}

sub1();
sub2();
sub1();
```

Dynamic scoping

Lexical variable

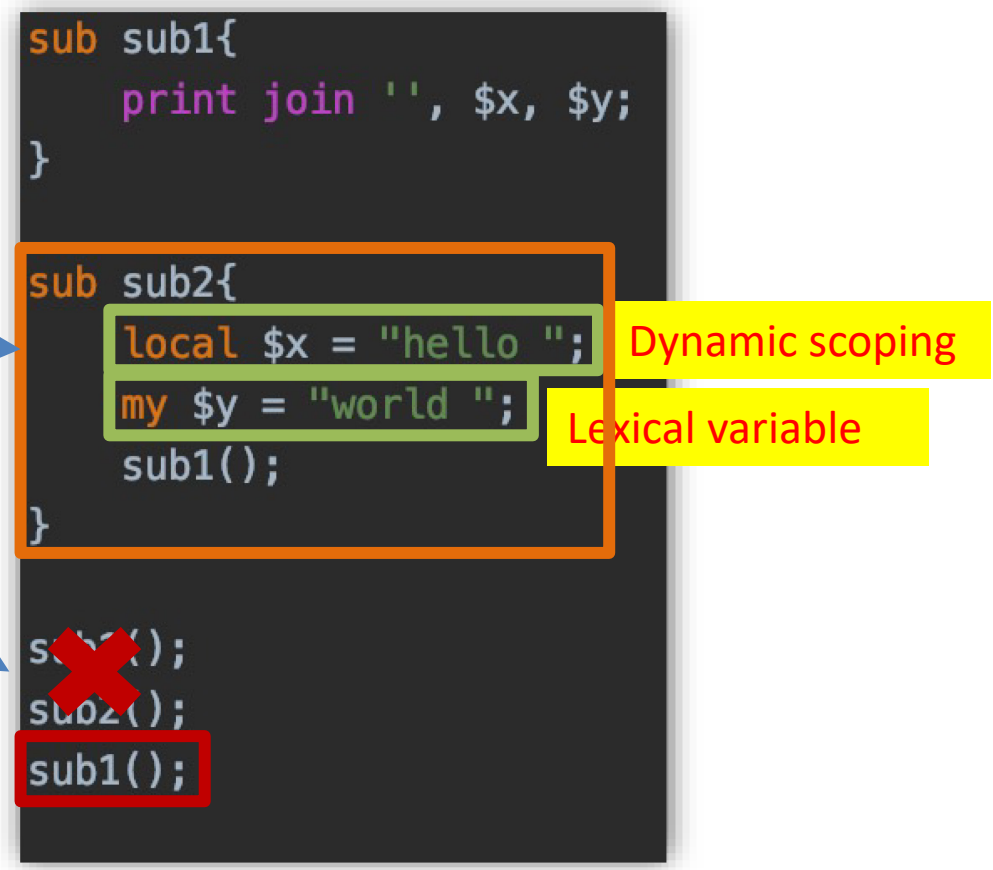
Package Variables: Example

- **An example with both static & dynamic scoping + Lexical variables**
  - No output since $x and $y are not defined
  - Output: “hello”
    - $y is a lexical variable of sub2
    - $y is not defined in sub1



Package Variables: Example

- **An example with both static & dynamic scoping + Lexical variables**
  - No output since $x and $y are not defined
  - Output: “hello”
  - No output since $x and $y is not defined
    - $x = “hello” only in sub2()
    - $y = “world” only in sub2()
    - sub2 is not an active subprogram!



Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - What if we define variable $x globally?

Dynamic scoping

Lexical variable

```
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{
    local $x = "Hello ";
    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - Output “Hi”
    - All functions have access to the $x

```
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{
    local $x = "Hello ";
    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - Output “Hi”
  - Output “Hello ”
    - Temporarily change $x to “Hello ”
    - sub1 can’t access $y (defined by sub2)

```
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{
    local $x = "Hello ";
    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - Output “Hi”
  - Output “Hello ”
  - Output “Hello world”
    - $y is defined in sub3

```perl
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{
    local $x = "Hello ";
    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - Output “Hi”
  - Output “Hello ”
  - Output “Hello world”
- What if we don’t modify $x?
  - “Hello world” or “Hi world”?

```
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{
    
    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Package Variables: Example

- **Another example with dynamic scoping & Lexical variables**
  - Output “Hi”
  - Output “Hello ”
  - Output “Hello world”
- What if we don’t modify $x?
  - “Hello world” or “Hi world”?
  - Output “Hi world”

```
$x = "Hi "; # our $x = "Hi ";

sub sub1{
    print join '', $x, $y;
}

sub sub2{
    local $x = "Hello ";
    my $y = "world";
    sub1();
}

sub sub3{

    my $y = "world";
    print join '', $x, $y;
}

sub1();
sub2();
sub3();
```

Conclusion

- **Dynamic Scoping**
  - Variables modified by local allow dynamic scoping, but my does not
  - Allows operators to change **temporarily** (and restore later)
  - Pros: Users can use variables from parent functions like parameters
  - Cons: Readability and comprehension of the code will be greatly reduced

Conclusion

➢ Define variables

- my => creates a local variable
- our => creates a package variable
- local => temporarily changes the local value of a global variable

➢ Suggestions

- Avoiding using dynamic scoping variables unless you actually need it
- It's always better to explicitly declare and use a package variable with its package name in Perl

Q&A