# Tutorial 6: Perl Introduction

Yanjing Ren & Huancheng Puyang

yjren22@cse.cuhk.edu.hk

hcpuyang22@cse.cuhk.edu.hk

# Outline

- Introduction of Perl
  - History, Introduction, Pros, Cons…
- Programming Environment
  - Command line, Editors, IDEs…
- Basic Properties & Grammar
  - Scalars, conditions, loops…
- Object Oriented Programming in Perl
  - Object, class, method, inheritance…

# Perl

- Perl programming language
  - an interpreted language
  - dynamic, general-purpose, and object oriented
- Special features
  - As powerful as C and as convenient as script description languages such as awk and sed
  - supports both static and dynamic scoping
  - supports regular expression and string passing

# Perl

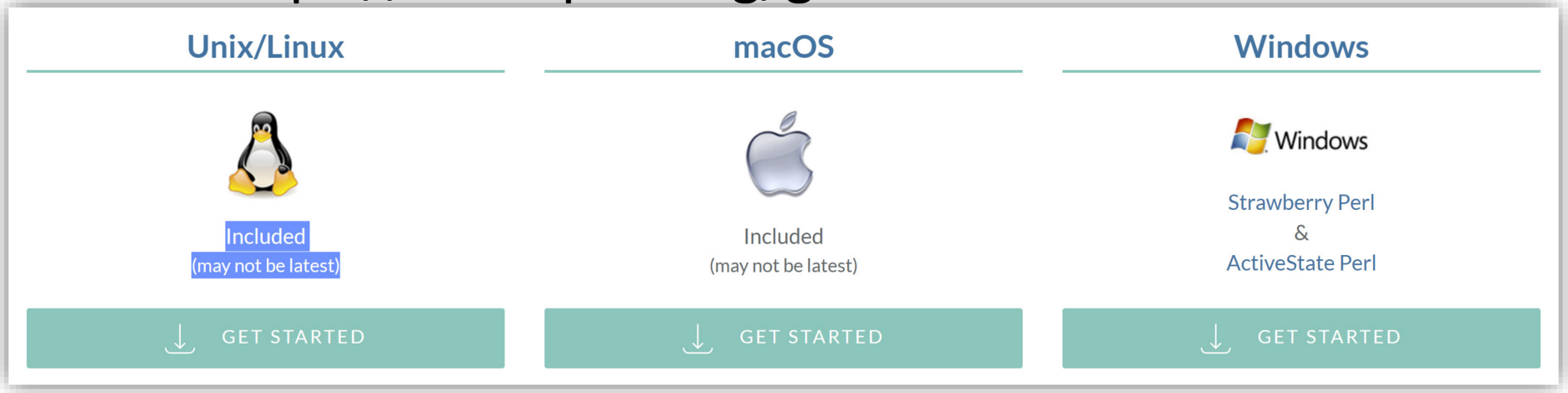
- Advantages
  - Directly provides more convenient programming elements such as generic variables, dynamic arrays, and Hash tables
  - Extensible, and we can find many of the modules we need through the CPAN ("the Comprehensive Perl Archive Network") central repository
- Disadvantages
  - Unreliability due to dynamic typing where variables are implicitly declared by simply assignments
    - You may get unexpected results due to typos (and there will be no errors reported)
  - Perl code may not be easy to read compared to some other languages
  - High memory usage

# Perl Environment

- Is it already installed? Which version?
  - perl –v

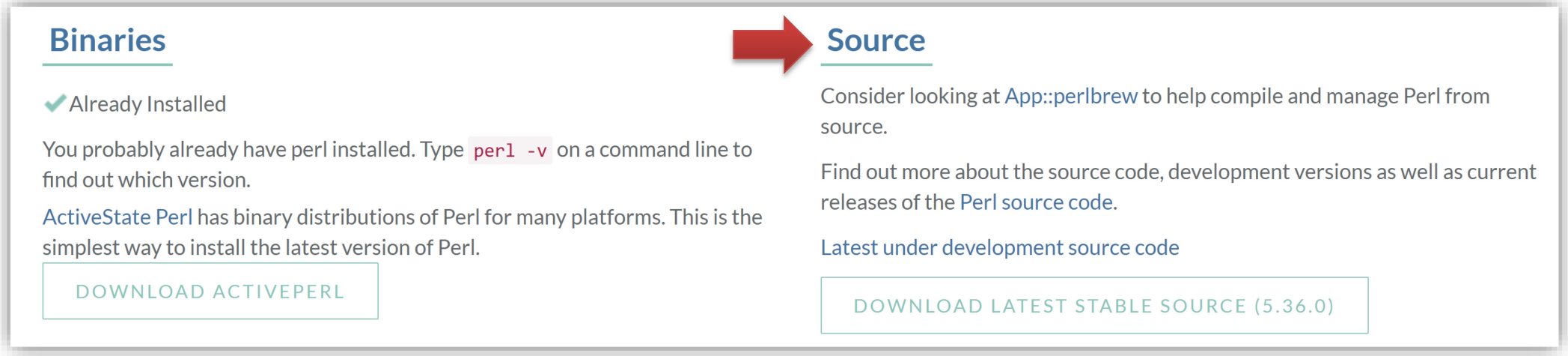
```
$ perl -v

This is perl 5, version 34, subversion 0 (v5.34.0) built for x86_64-linux-gnu-thread-multi
(with 57 registered patches, see perl -V for more detail)

Copyright 1987-2021, Larry Wall

Perl may be copied only under the terms of either the Artistic License or the
GNU General Public License, which may be found in the Perl 5 source kit.

Complete documentation for Perl, including FAQ lists, should be found on
this system using "man perl" or "perldoc perl".  If you have access to the
Internet, point your browser at http://www.perl.org/, the Perl Home Page.
```

- Download: https://www.perl.org/get.html



# Perl Environment

- Install on Unix/Linux

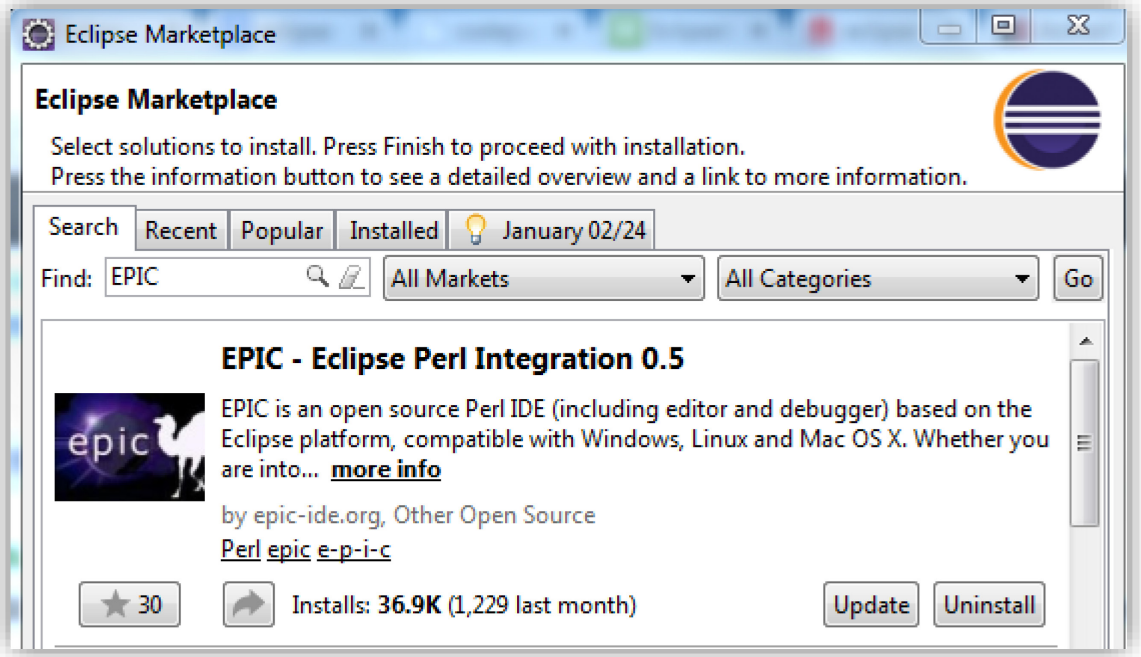



- Download source to avoid GitHub sign in
  - Compile and install:

```
$ tar -xzf perl-5.x.y.tar.gz
$ cd perl-5.x.y
$ ./Configure -de
$ make
$ make test
$ make install
```

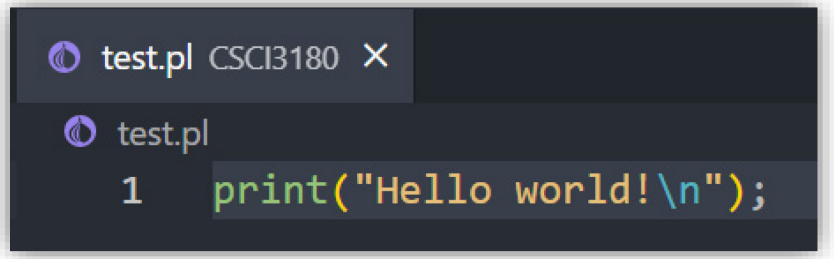
# Perl Environment

- Editors: vs-code, sublime, vim, emacs…
- IDEs
  - Padre: https://padre.perlide.org/
  - EPIC plugin for Eclipse: http://www.epic-ide.org/



# Running Your First Perl Program

- Using vs-code as an example
  - Create a Perl program file as "xxx**.pl**" (e.g., test.pl)




  - Run with terminal
    - Windows: use command prompt
    - Mac/Linux/Unix: Use terminal

```
# tinoryj @ Tinoryj-Laptop in ~/Projects/CSCI3180
$ perl test.pl
Hello world!
```

# Perl Variables

- Perl is a dynamic typed programming language
- 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

# Perl: Static and Dynamic Scoping

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

```
$a = 0;
sub foo {
  return $a;
}
sub staticScope {
  my $a = 1; # lexical (static)
  return foo();
}

print staticScope(); # 0 (from the saved global frame)

$b = 0;
sub bar {
  return $b;
}
sub dynamicScope {
  local $b = 1;
  return bar();
}
print dynamicScope(); # 1 (from the caller's frame)
```

# Perl Variables: Scalar

- Scalar
  - preceded by a dollar sign $
  - store either a string, a number or a reference

test.pl CSCI3180

test.pl

```
1  $age = 25;
2  $name = "runoob";
3  $salary = 1445.50;
4  print "Age = $age\n";
5  print "Name = $name\n";
6  print "Salary = $salary\n";
```

```
$ perl test.pl
Age = 25
Name = runoob
Salary = 1445.5
```

# Perl Variables: Array

- Array
  - preceded by sign @
  - store ordered lists of scalars

test.pl CSCI3180

test.pl

```
1  @ages = (25, 30, 40);
2  print "\$ages[0] = $ages[0]\n";
3  print "\$ages[1] = $ages[1]\n";
4  print "\$ages[2] = $ages[2]\n";
```

```
$ perl test.pl
$ages[0] = 25
$ages[1] = 30
$ages[2] = 40
```

# Perl Variables: Hash

- Hash
  - preceded by sign %
  - store sets of key/value pairs

test.pl CSCI3180

test.pl

```
1  %people = ("name" => "CSCI", "age" => "30");
2  print $people{"name"};
3  print "\n";
4  print $people{"age"};
```

```
$ perl test.pl
CSCI
30%
```

  - In some other language, it's called dictionary.

# Perl: References and Dereference

- References
  - Put a \ in front of a variable

```
$aref = \@array;          # $aref now holds a reference to @array
$href = \%hash;           # $href now holds a reference to %hash
$sref = \$scalar;         # $sref now holds a reference to $scalar
```

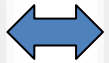
  - For the reference, you can copy it or store it

```
my $xy = $aref;           # $xy now holds a reference to @array
my @p;
$p[3] = $href;            # $p[3] now holds a reference to %hash
$z = $p[3];               # $z now holds a reference to %hash
```

# Perl: References and Dereference

- References
  - [ ITEMS ] makes a new, anonymous array, and returns a reference to that array

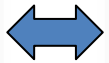
```
$aref = [ 1, "foo", undef, 13 ];
# $aref now holds a reference to an array
```



```
@a = (1, "foo", under, 13);
```

  - { ITEMS } makes a new, anonymous hash, and returns a reference to that hash

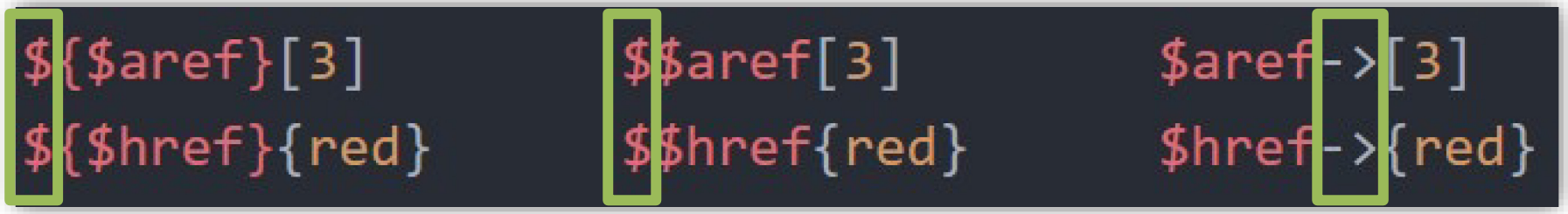
```
$href = { APR => 4, AUG => 8 };
# $href now holds a reference to a hash
```



```
%h = ( APR => 4, AUG => 8 );
```

# Perl: References and Dereference

- Access to the references
  - Note, the references in Perl are actually pointers
    - We access/change variables using references by:




- Deference
  - The most direct way of dereferencing a reference is to prepend the corresponding data type character that you are expecting in front of the scalar variable containing the reference.

Create reference->

```
@array = ('1', '2', '3');
$reference_array = \@array;
print @$reference_array;
%hash = ('1'=>'a', '2'=> 'b', '3' => 'c');
$reference_hash = \%hash;
print %$reference_hash;      # Dereferencing
$scalar = 1234;
$reference_scalar = \$scalar;
print $$reference_scalar;    # Dereferencing
```

<-Dereference

# Perl Grammar: For Loop

- Traditional numerical for loop
  - E.g., print the numbers from 0 through 4

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

- For loop through an array

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

- For loop through an hash

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

# Perl Grammar: If Statement

- Perl if statement allows you to control the execution of your code based on conditions

```
my $a = 1;
print("Welcome to Perl if tutorial\n") if($a == 1);
```

<-Exec if first

- In some cases, you want to also execute another code block if the expression does not evaluate to true

```
my $a = 1;
my $b = 2;
if($a == $b){
 print("a and b are equal\n");
}else{
 print("a and b are not equal\n");
}
```

```
my $a = 1;
my $b = 2;

if($a == $b){
   print("a and b are equal\n");
}elsif($a > $b){
   print("a is greater than b\n");
}else{
   print("a is less than b\n");
}
```

- Perl provides the if-elsif statement for checking multiple conditions and executing the corresponding code block

# Perl Grammar: Functions or Subroutines

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

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

my $total = sum(1, 2);
print "\n$total\n";
```

```perl
my $x = shift;
```

```perl
my $y = shift;
```

```perl
return EXPR;
```

Results

```
$ perl test.pl
1 2
3
```

# Perl: Packages and Modules

- Package
  - A collection of code which lives in its own namespace
  - Explicitly refer to functions and variables in a package using "::"
  - Prevent variable name collisions between packages

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

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

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

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

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

# Object Oriented Programming (OOP)

- Object
  - A reference to a data type that knows what class it belongs to
  - **bless** a reference as an 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

# Object Oriented Programming (OOP)

- Defining a Class
  - A class corresponds to a Package
  - A package is a self-contained unit of user-defined variables and subroutines
- To declare a class named “ClassSelection”

```
package ClassSelection;

sub new
{
    my $className = shift;
    my $classInfo = {
        '_classID' => shift,
        '_classTutor' => shift,
    };
    bless $classInfo, $className;
    return $classInfo;
}
```

- Creating and Using Objects
  - Constructor & Object (instance)
    - create a constructor for Person class using a hash reference
    - The object reference is created by blessing a reference to the package’s class

Or

```
my $object = ClassSelection->new("CSCI3180","Wallace");
```

```
my $object = new ClassSelection("CSCI4180", "Wallace");
```

# Object Oriented Programming (OOP)

- Defining Methods

```
sub setClassID {
    my ( $classInfo, $classID ) = @_;
    $classInfo->{_classID} = $classID;
    return $classInfo->{_classID};
}

sub getClassID {
    my ( $classInfo ) = @_;
    return $classInfo->{_classID};
}
```

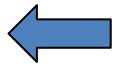
- Putting these codes in a file as a module
  - Loading the module needs to see a true statement there

```
package ClassSelection;

sub new
{
    my $className = shift;
    my $classInfo = {
        '_classID' => shift,
        '_classTutor' => shift,
    };
    bless $classInfo, $className;
    return $classInfo;
}

sub setClassID {
    my ( $classInfo, $classID ) = @_;
    $classInfo->{_classID} = $classID;
    return $classInfo->{_classID};
}

sub getClassID {
    my ( $classInfo ) = @_;
    return $classInfo->{_classID};
}

1;
```

# Object Oriented Programming (OOP)

- Using Person class (ClassSelection.pm) from main.pl
  - Store class in “xx.pm” file
  - Load in other Perl program

```
$ perl main.pl
Original class ID is : CSCI-3180
New class ID is : ENGG-3180
```



```perl
require "./ClassSelection.pm";

my $object = ClassSelection->new("CSCI-3180", "Wallace");
my $classID = $object->getClassID();
print "Original class ID is : $classID\n";

$object->setClassID( "ENGG-3180" );
$classID = $object->getClassID();
print "New class ID is : $classID\n";
```

- Inheritance
  - When a class inherits from another class, any methods defined in the parent class are available to the child class

```perl
require "./ClassSelection.pm";
package ClassSelectionForCSE;
use strict;
our @ISA = qw(ClassSelection); # inherits from ClassSelection
```

# Object Oriented Programming (OOP)

```perl
require "./ClassSelectionForCSE.pm";

my $object = ClassSelectionForCSE->new("CSCI-3180", "Wallace");
my $classID = $object->getClassID();
print "Original class ID is : $classID\n";

$object->setClassID( "ENGG-3180" );
$classID = $object->getClassID();
print "New class ID is : $classID\n";
```

```perl
require "./ClassSelection.pm";

my $object = ClassSelection->new("CSCI-3180", "Wallace");
my $classID = $object->getClassID();
print "Original class ID is : $classID\n";

$object->setClassID( "ENGG-3180" );
$classID = $object->getClassID();
print "New class ID is : $classID\n";
```

```
$ perl main.pl
Original class ID is : CSCI-3180
New class ID is : ENGG-3180
```

# Note: Useful Functions

- Function call
  - You can call function with or without ()

```
print "Hello world!\n";
print ("Hello world!\n");
```

```
$ perl test.pl
Hello world!
Hello world!
```

- 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
```

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

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

# Note: 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

```
1  use strict;
2  use warnings;
3
4  #rest of your program
```

# Learning Resources

- Perl tutorial website
  - https://www.tutorialspoint.com/perl/index.htm
- Install Perl modules
  - http://www.cpan.org/modules/index.html

# Q&A