Special Variables

Spoken Tutorial Project
http://spoken-tutorial.org
National Mission on Education through ICT
http://sakshat.ac.in

Nirmala Venkat

27 May 2015

Learning Objective

We will learn about

Learning Objective

We will learn about

- Global special variables

Learning Objective

We will learn about

- Global special variables
- Special command line variables

Learning Objective

We will learn about

- Global special variables
- Special command line variables
- Global special constants

System Requirements

System Requirements

- **Ubuntu Linux 12.04 OS**

System Requirements

- **Ubuntu Linux 12.04 OS**
- Perl 5.14.2

System Requirements

- **Ubuntu Linux 12.04 OS**
- **Perl 5.14.2**
- **gedit Text Editor**

Pre-requisites

- **Working knowledge of Perl Programming**

Pre-requisites

- **Working knowledge of Perl Programming**
- **For relevant Perl tutorials, visit http://spoken-tutorial.org**

What are special variables?

What are special variables?

- **Special variables are predefined variables, that have a special meaning in Perl**

What are special variables?

- **Special variables are predefined variables, that have a special meaning in Perl**
- **Do not need to be initialised before use**

What are special variables?

- **Special variables are predefined variables, that have a special meaning in Perl**
- **Do not need to be initialised before use**
- **Used to hold the results of searches, environment variables, flags to control debugging**

Global special variables

Global special variables

$_ - Implicit variable

- Is a widely used special variable

Global special variables

$_ - Implicit variable

- **Is a widely used special variable**
- **Default parameter for lot of functions and pattern-searching string**

Global special variables

`@_`

- Is used to store subroutine parameters

Global special variables

@_

- **Is used to store subroutine parameters**
- **Arguments for a subroutine are stored in this array variable**

Global special variables

@_

- Is used to store subroutine parameters
- Arguments for a subroutine are stored in this array variable
- Array operations like pop/shift can be done on this variable

Global special variables

%ENV

- Environment variables contain a copy of the current environment variables, such as

  PWD

  USER

  LANG

  PATH etc.

Global special variables

$0

- Contains the name of the current Perl program that is being executed

Global special variables

$0

- **Contains the name of the current Perl program that is being executed**
- **Generally used for logging purpose**

Global special variables

$0

- Contains the name of the current Perl program that is being executed
- Generally used for logging purpose

Filename: First.pl

Global special variables

$0

- Contains the name of the current Perl program that is being executed
- Generally used for logging purpose

Filename: First.pl

Example: `print $0;`

Global special variables

$0

- Contains the name of the current Perl program that is being executed
- Generally used for logging purpose

Filename: First.pl

Example: `print $0;`

Output: First.pl

Global special variables

<=> - Sort comparison variable

- Perl has a built-in function called **sort** that sorts an array

Global special variables

<=> - Sort comparison variable

- Perl has a built-in function called sort that sorts an array
- A comparison function will compare its parameters using the <=>

Global special variables

$!

- If used in string context, it returns the system error string

Global special variables

$!

- **If used in string context, it returns the system error string**
- **Example:**

```
open FH <hello.txt or die ”Cannot
open file for reading : $!”;
```

Global special variables

$!

- If used in string context, it returns the system error string
- **Example:**
  open FH <hello.txt or die "Cannot open file for reading : $!";
- If the file **hello.txt** doesn't exist, it will print the error message

Global special variables

$@

Global special variables

$@

- **It returns error message, returned from eval or require command**

Global special variables

$@

- It returns error message, returned from **eval** or **require** command
- **Example:**

```
my $result = eval {$x/ $y};
print "could not divide $@" if $@
```

Global special variables

$$

- This holds the process ID of the Perl interpreter running this script

Global special variables

$$

- This holds the process ID of the Perl interpreter running this script
- Example:

```
print "$$";
```

Output:

```
17648
```

Special command line variables

<>

- The **diamond operator** is used to read every line, from the files specified on the command line

Special command line variables

<>

- **The diamond operator is used to read every line, from the files specified on the command line**
- **Example:**

```
while(<>) {
print "$_ \n "; }
```

Special command line variables

@ARGV

Special command line variables

@ARGV

- Holds all the values from the command line

Special command line variables

@ARGV

- **Holds all the values from the command line**
- **No need to declare the variables**

Special command line variables

@ARGV

- **Holds all the values from the command line**
- **No need to declare the variables**
- **Example:**

```
foreach(@ARGV) {
print;
print ”\n ”; }
```

Global Special constants

- **__END__ : Indicates the logical end of the program**

Global Special constants

- **__END__ : Indicates the logical end of the program**
- **__FILE__ : Represents the filename of the program**

Global Special constants

- **__END__ : Indicates the logical end of the program**
- **__FILE__ : Represents the filename of the program**
- **__LINE__ : Represents the current line number**

Global Special constants

- **__END__ : Indicates the logical end of the program**
- **__FILE__ : Represents the filename of the program**
- **__LINE__ : Represents the current line number**
- **__PACKAGE__ : Represents the current package name at compile time**

Summary

In this tutorial, we learnt about

- some commonly used special variables in Perl

Assignment

1. Write a Perl script to sort the following array of numbers in ascending and descending order.
   my @numbers = (22, 88, 33, 55, 11);
2. Note: For descending order, use the below code for comparison
   Sort{ $b <=>$a} @numbers;

Assignment (cont.)

3. Print the sorted result using **while loop** and special variable **$_**
4. Save and execute the program
5. Check your result

About the Spoken Tutorial Project

- **Watch the video available at http://spoken-tutorial.org /What_is_a_Spoken_Tutorial**
- **It summarises the Spoken Tutorial project**
- **If you do not have good bandwidth, you can download and watch it**

Spoken Tutorial Workshops

The Spoken Tutorial Project Team

- **Conducts workshops using spoken tutorials**
- **Gives certificates to those who pass an online test**
- **For more details, please write to contact@spoken-tutorial.org**

Acknowledgements

- Spoken Tutorial Project is a part of the Talk to a Teacher project
- It is supported by the National Mission on Education through ICT, MHRD, Government of India
- More information on this Mission is available at http://spoken-tutorial.org/NMEICT-Intro