

# **Introduction to Perl6**

**Gabor Szabo**

## **Introduction to Perl6**

by Gabor Szabo

0.04 Edition

Published Sat May 2 12:30:51 2009

Copyright © 2006, 2007, 2008, 2009 PTI Ltd. Perl Training Israel <http://www.pti.co.il/>

# Table of Contents

<b>1. About Perl Training Israel</b> .....	<b>1</b>
<b>2. Introduction</b> .....	<b>2</b>
2.1. Self Introduction - Who am I ? .....	2
2.2. Self Introduction - Who are you ?.....	2
<b>3. Virtual Box setup</b> .....	<b>3</b>
<b>4. Introduction to Perl 6</b> .....	<b>4</b>
4.1. Disclaimer .....	4
4.2. Getting started .....	4
4.3. Other resources .....	4
4.4. Installing Rakudo .....	5
4.5. Development Environment.....	5
4.6. Running Rakudo .....	6
4.7. Hello World.....	6
4.8. Comments .....	7
4.9. POD - Plain Old Documentation .....	7
<b>5. First steps in Perl6</b> .....	<b>9</b>
5.1. Hello World - scalar variables.....	9
5.2. Hello World - interpolation .....	9
5.3. Reading from the keyboard.....	10
5.4. Numerical operations .....	10
5.5. Add.....	11
5.6. String operators .....	11
5.7. String repetition.....	12
5.8. if statement - comparing values .....	12
5.9. Naked if statement.....	13
5.10. Other forms of the if statement .....	13
5.11. Ternary Operator .....	13
5.12. Comparison Operators .....	14
5.13. Boolean expressions (logical operators) .....	15
5.14. Chained comparisons .....	16
5.15. Comparing values - Calculator.....	17
5.16. Calculator - given.....	17
5.17. String functions: index .....	18
5.18. String functions: substr .....	18
5.19. Super-or (junctions) .....	18
<b>6. Files in Perl6</b> .....	<b>20</b>
6.1. exit, warn, die.....	20
6.2. Read line from file.....	20
6.3. Process a file line by line.....	21
6.4. Write to a file.....	21
6.5. Open file modes .....	22
6.6. Exercise: Print sum of numbers .....	22
6.7. Solution .....	22
6.8. undef, defined, not.....	23

<b>7. Perl6 Lists and Arrays.....</b>	<b>24</b>
7.1. List Literals, list ranges .....	24
7.2. List Assignment .....	24
7.3. Swap two values.....	24
7.4. loop over elements of list with for .....	25
7.5. Create array, go over elements .....	25
7.6. Array elements (create menu).....	26
7.7. Array assignment .....	26
7.8. Command line options .....	27
7.9. Process CSV file.....	27
7.10. join .....	28
7.11. Hyper Operators .....	28
7.12. slurp.....	29
7.13. xx - string multiplicator.....	29
<b>8. Perl6 Hashes .....</b>	<b>30</b>
8.1. Associative Arrays (Hashes) .....	30
8.2. Fetching data from a hash .....	30
8.3. Multidimensional hashes.....	30
8.4. Count words .....	31
8.5. Overview .....	31
8.6. slurp.....	32
8.7. kv.....	32
<b>9. Subroutines in Perl6 .....</b>	<b>34</b>
9.1. Simple definition with required parameters .....	34
9.2. Named parameters.....	34
9.3. Subroutines.....	35
9.4. No parameter definition - perl 5 style .....	36
9.5. Subroutines.....	37
9.6. Subroutines.....	37
<b>10. Perl 6 Regexes.....</b>	<b>39</b>
10.1. Regexes in Perl 6.....	39
10.2. Simple matching .....	39
10.3. Substitute.....	40
<b>11. Junctions in Perl6.....</b>	<b>41</b>
11.1. Junctions.....	41
11.2. More examples with Junctions.....	41
<b>12. Perl5 to Perl6.....</b>	<b>44</b>
12.1. Intro .....	44
12.2. Hello World.....	44
12.3. Scalars .....	44
12.4. Arrays.....	45
12.5. Hashes .....	46
12.6. Control Structures .....	46
12.7. Functions .....	47
12.8. Files.....	48
12.9. Modules, Classes.....	48

12.10. Perl 5 to Perl6.....	49
<b>13. Shell to Perl6 .....</b>	<b>50</b>
13.1. Intro.....	50
13.2. Unix commands in Perl6.....	50
13.3. awk.....	50
13.4. cat.....	50
13.5. cd in Perl6 .....	51
13.6. chmod.....	51
13.7. chown.....	51
13.8. cmp.....	51
13.9. compress.....	51
13.10. cut.....	51
13.11. date.....	51
13.12. diff.....	52
13.13. df.....	52
13.14. dos2unix.....	52
13.15. du.....	52
13.16. file.....	52
13.17. find.....	52
13.18. grep.....	52
13.19. gzip.....	53
13.20. head.....	53
13.21. kill.....	53
13.22. ln.....	53
13.23. ls.....	53
13.24. mkdir.....	53
13.25. mv.....	53
13.26. ps.....	53
13.27. popd.....	53
13.28. pushd.....	54
13.29. pwd.....	54
13.30. rmdir.....	54
13.31. rm.....	54
13.32. sed.....	54
13.33. sort.....	54
13.34. tail.....	54
13.35. tar.....	55
13.36. touch.....	55
13.37. uniq.....	55
13.38. unix2dos.....	55
13.39. wc.....	55
13.40. who.....	56
13.41. zip.....	56
13.42. Other UNIX command.....	56

<b>14. Object Oriented Perl6 .....</b>	<b>57</b>
14.1. Simple Point class .....	57
14.2. Read/write attributes and accessors .....	57
14.3. Class Methods .....	58
14.4. Private Attributes.....	58
14.5. Method with parameters.....	59
14.6. Inheritance of classes .....	59
14.7. Classes in Perl 6 .....	60
<b>15. Old slides.....</b>	<b>62</b>
15.1. Why old slides?.....	62
15.2. Installing Pugs on Linux .....	62
15.3. Building Pugs on Ubuntu .....	62
15.4. Running Pugs .....	63
15.5. Division .....	63
<b>Table of Contents .....</b>	<b>65</b>

# List of Tables

5-1. Comparison Operators.....14  
5-2. Logical operators.....15

# Chapter 1. About Perl Training Israel

PTI has been providing training and development services since 2000. Our courses include the following:

## *Perl Courses*

- Fundamentals of Perl
- References, Modules and Objects
- Web Application Development with Perl
- Debugging Perl Scripts and Applications
- QA Test Automation using Perl
- Perl 6 for programmers

## *Non-Perl Courses*

- Regular Expressions
- Version Control with Subversion

Details can be found on our web site <http://www.pti.co.il>

# Chapter 2. Introduction

## 2.1. Self Introduction - Who am I ?

- Gabor Szabo <gabor@pti.co.il> <http://www.pti.co.il>
- First program in 1983
- In the field since 1993
- Perl trainer since 2000
- Perl developer
- CPAN author
- System Administrator
- Web application developer
- Testing Automation
- <http://www.szabgab.com>

## 2.2. Self Introduction - Who are you ?

- Name
- Workplace
- Background in computers (e.g. Operating Systems)
- Background in programming (which languages? How many years?)
- Background in Perl?
- Why do you take this course ?
- What do you expect to accomplish ?

# Chapter 3. Virtual Box setup

In order to setup the Virtual Box image please follow the instructions:

## *Add disks*

- Copy Foobar\_\*.vdi to ~/.VirtualBox/VDI/ (Optional)
- Launch Virtual Box
- File/Virtual Disk Manager
- Add
- Select Foobar\_root.vdi
- Add
- Select Foobar\_usr.vdi

## *Setup Machine*

- Machine/New
- Next
- Name: Foobar, OS Type: Ubuntu; Next
- Base Memory: 512 Mb; Next
- Boot Hard disk: Select Foobar\_root.vdi; Next
- Finish
- Select the icon of the newly created Foobar machine
- Machine/Settings
- Select Hard disks
- Click on Add Attachment (+ sign on the right)
- Select Foobar\_usr.vdi; OK
- Machine/Start

# Chapter 4. Introduction to Perl 6

## 4.1. Disclaimer

Perl 6 is not fully specified yet

Even what is specified is not fully implemented yet

I hardly know any Perl 6

This is my first experiment in creating training material for Perl 6.

Some say that Perl 6 provides many new ways to shoot yourself in the leg.

I am just trying to help in this. ;-)

Be WARNED.

## 4.2. Getting started

- There are several partial implementation of Perl 6.
- Pugs (written in Haskell) was the first usable but it is not maintained now.
- Rakudo is running on Parrot and it is currently the most promising implementaton.
- SMOP - Simple Meta Object Programming / Simple Matter Of Programming
- <http://www.perlfoundation.org/perl6/index.cgi?smop>
- NQP - Not Quite Perl
- kp6 - KindaPerl6
- Other implementations: [http://www.perlfoundation.org/perl6/index.cgi?perl\\_6\\_implementations](http://www.perlfoundation.org/perl6/index.cgi?perl_6_implementations)
- The Specs can be found in the Pugs SVN repository: <http://svn.pugscode.org/pugs/docs/Perl6/Spec/>
- The Specs Tests too: <http://svn.pugscode.org/pugs/t/spec>
- The Specs with tests: <http://perlcabal.org/syn/>

## 4.3. Other resources

- A German intro: <http://wiki.perl-community.de/bin/view/Wissensbasis/PerlTafel>
- The tutorial blogs of Moritz Lenz <http://perlgeek.de/blog-en/perl-5-to-6/>
- Rakudo: <http://rakudo.org/>
- [http://www.athenalab.com/Perl\\_6\\_Users\\_FAQ.htm](http://www.athenalab.com/Perl_6_Users_FAQ.htm)
- Mailing list for Perl 6 early users <mailto:perl6-users-subscribe@perl.org>
- <http://www.programmersheaven.com/2/Perl6-FAQ>
- Feather <http://feather.perl6.nl/>
- Examples in the source tree of Pugs: <http://svn.pugscode.org/pugs/examples/>
- Perl 6 Grammars by Chris Dolan: <http://chrisdolan.net/frozenperl/rakudo-grammars.html>

## 4.4. Installing Rakudo

\* *While there are monthly releases of Parrot and separately releases of Rakudo but you cannot really "install" them yet. Besides their development is still so fast that I'd recommend using the latest as checked out from their respective version controls system. That's what I am going to describe in the next slides.*

\* *In the first version I'll describe the method for Linux and later for Windows as well*

Currently the recommended way to install Rakudo is the following:

For this you'll need to have Subversion and Git installed:

```
$ cd ~
$ mkdir somedir
$ cd somedir
$ git clone git://github.com/rakudo/rakudo.git
$ cd rakudo
$ perl Configure.pl --gen-parrot
$ make
$ make test
$ make spectest
```

For up-to-date instructions, please visit the Rakudo web site at <http://rakudo.org/>

## 4.5. Development Environment

Any text editor.

**Padre** <http://padre.perlide.org/> with the Perl6 plugin: <http://search.cpan.org/dist/Padre-Plugin-Perl6>

- \* Set the `PARROT_DIR` variable to point to the checked out directory of Parrot and a `RAKUDO_DIR` variable to point to the checked out directory of Rakudo.

**vi** with syntax highlighting using `utils/perl6.vim` from the Pugs source tree

- \* The `util/perl6.vim` file in the Pugs source tree. copy that file to `~/vim/syntax/` or create a symbolic link. In `~/vimrc` include the following line: `autocmd BufNewFile,BufRead *.p6 setf perl6` or put these lines in `~/vim/filetype:` `augroup filetypeperl6 autocmd BufNewFile,BufRead *.p6 setf perl6 augroup END`

**emacs** with perl6 mode using `utils/cperl-mode.el` from the Pugs repository.

## 4.6. Running Rakudo

Version information:

```
/path/to/somedir/rakudo/perl6 -v
```

I addedd `/path/to/somedir/rakudo/` to the `PATH` variable so I can also type:

```
perl6 -v
```

Oneliners:

```
perl6 -e "say 42"
```

Help:

```
perl6 -h
```

Interactive perl6 console

```
perl6
```

- \* I used to have a shell script called `perl6` that contains the following: `/path/to/somedir/parrot/parrot /path/to/somedir/parrot/parrot/languages/rakudo/perl6.pbc "$@"` but recently I am just using the generated `perl6` executable. It is said to be crashing sometimes but it worked quite well for me so far.

## 4.7. Hello World

One usually starts by saying something to the audience.

We do this using the `say` keyword.

Not only does it print to the screen it also adds a newline at the end of the string.

As you can see strings are marked by `"` at both their ends.

Commands end with `;`

**Example 4-1. examples/intro/hello\_world.p6**

```
#!/usr/bin/perl6
use v6;

say "Hello World";
```

The same in OOP style:

**Example 4-2. examples/intro/hello\_world\_oop.p6**

```
#!/usr/bin/perl6
use v6;

"Hello World".say;
```

## 4.8. Comments

**Example 4-3. examples/intro/comments.p6**

```
#!/usr/bin/perl6
use v6;

say "Hello";

# This is a single line comment

say #< embedded comments > "hello world";

#[
  This is a
  multiline
  comment
  It starts by a # followed by an opening bracket
  and end with the bracket pair
]

#<<<
  but it has a bug now
  that it cannot start on the first character of a line
>>>
```

See also Synopsis 2 - Whitespace and Comments

## 4.9. POD - Plain Old Documentation

### Example 4-4. examples/intro/documentation.p6

```
#!/usr/bin/perl6
use v6;

print "Hello";

=begin pod

=head1 Title

text

=end pod

say " World";
```

See also the S26 documentation which was written using the Perl 6 pods

# Chapter 5. First steps in Perl6

## 5.1. Hello World - scalar variables

Strings can be stored in so called scalar variables.

Each such variable starts with a \$ sign and then alphanumeric characters and underscore.

Before using such a variable the first time one has to declare it by the *my* keyword.

```
my $this_is_a_variable;  
my $ThisIsAnotherVariableButWeDontLikeItSoMuch;
```

Variables are case sensitive.

```
my $h;  
my $H;
```

### Example 5-1. examples/scalars/hello\_world\_variable.p6

```
#!/usr/bin/perl6  
use v6;  
  
my $greeting = "Hello World";  
say $greeting;
```

By default scalar variables have no specific types but later we will see how can we restrict the values a scalar can hold to an Int or Str or any other type.

One can also declare a variable without giving it an initial value:

```
my $x;
```

In such case the value of the variable will be **undef**.

## 5.2. Hello World - interpolation

We can put scalar variables within regular strings. Their value will be interpolated into the string.

**Example 5-2. examples/scalars/hello\_world\_variable\_interpolation.p6**

```
#!/usr/bin/perl6
use v6;

my $name      = "Foo";
my $greeting = "Hello $name";
say $greeting;
```

## 5.3. Reading from the keyboard

Ask the user what's her name

When prompting the user with a question it is probably the best to use the *prompt* function. Similarly to *say* it prints to the screen but without the newline at the end, waits for the user to type in something.

It reads up to the ENTER the user presses but passes over only the the part before the newline. (Perl 5 users could think about it as having `autochomp`)

**Example 5-3. examples/scalars/read\_stdin.p6**

```
#!/usr/bin/perl6
use v6;

my $name = prompt("Please type in yourname: ");

say "Hello $name";
```

## 5.4. Numerical operations

Numerical operator can be used on the scalar values.

**Example 5-4. examples/scalars/numerical\_operators.p6**

```
#!/usr/bin/perl6
use v6;

my $x = 3;
my $y = 11;

my $z = $x + $y;
say $z;           # 14
```

```

$z = $x * $y;
say $z;           # 33
say $y / $x;     # 3.666666666666667

$z = $y % $x;    # (modulus)
say $z;          # 2

$z += 14;        # is the same as   $z = $z + 14;
say $z;          # 16

$z++;           # is the same as   $z = $z + 1;
$z--;           # is the same as   $z = $z - 1;

$z = 23 ** 2;   # exponentiation
say $z;         # 529

```

## 5.5. Add

As we can see - in this case - Perl 6 does not care that you typed in a string, the numeric + adds together the two numbers.

### Example 5-5. examples/scalars/add.p6

```

#!/usr/bin/perl6
use v6;

my $a = prompt "First number:";
my $b = prompt "Second number:";

my $c = $a + $b;

say "\nResult: $c";

```

## 5.6. String operators

### Example 5-6. examples/scalars/string\_operators.p6

```

#!/usr/bin/perl6
use v6;

my $x = "Hello";
my $y = "World";

# ~ is the concatenation operator, attaching one string after the other
my $z = $x ~ " " ~ $y; #           the same as "$x $y"
say $z;                # Hello World

```

```

my $w = "Take " ~ (2 + 3);
say $w;                # Take 5
say "Take 2 + 3";      # Take 2 + 3
say "Take {2 + 3}";    # Take 5

$z ~= "! ";           # the same as $z = $z ~ "! ";
say "$z";              # 'Hello World! '

```

~ concatenates two strings.

#### Example 5-7. examples/scalars/concat.p6

```

#!/usr/bin/perl6
use v6;

my $a = prompt "First string:";
my $b = prompt "Second string:";

my $c = $a ~ $b;

say "\nResult: $c";

```

\* As it can be seen from the above example any operation can be interpolated into a string using the curly brace syntax.

## 5.7. String repetition

#### Example 5-8. examples/scalars/string\_repetition.p6

```

#!/usr/bin/perl6
use v6;

my $z = "Hello World! ";

# x is the string repetition operator
my $q = $z x 3;
say "$q";          # 'Hello World! Hello World! Hello World! '

```

## 5.8. if statement - comparing values

You can compare two values or scalar variables using the *if* statement and one of the comparison operators.

**Example 5-9. examples/scalars/if.p6**

```
#!/usr/bin/perl6
use v6;

my $age = 23;
if ($age > 18) {
    say "You can vote in most countries.";
}
```

## 5.9. Naked if statement

**Example 5-10. examples/scalars/naked\_if.p6**

```
#!/usr/bin/perl6
use v6;

my $age = 23;
if $age > 18 {
    say "You can vote in most countries.";
}
```

## 5.10. Other forms of the if statement

```
if COND {
}
```

```
if COND {
} else {
}
```

```
if COND {
} elsif COND {
} elsif COND {
} else {
}
```

## 5.11. Ternary Operator

**Example 5-11. examples/scalars/ternary.p6**

```
use v6;
```

```

my $age = 42;

if ($age > 18) {
    say "Above 18";
} else {
    say "Below 18";
}

say $age > 18 ?? "Above 18" !! "Below 18";

COND ?? VALUE_IF_TRUE !! VALUE_IF_FALSE

```

## 5.12. Comparison Operators

Two sets of relation operators. One is to compare numerically the other is to compare as strings, based on the ASCII table.

**Table 5-1. Comparison Operators**

Numeric	String (ASCII)	Meaning
==	eq	equal
!=	ne	not equal
<	lt	less than
>	gt	greater than
<=	le	less than or equal
>=	ge	greater then or equal

See also S03-operators.pod

```

3 == 4           # false
'35' eq 35.0    # false
'35' == 35.0    # true
13 > 2          # true
13 gt 2         # false !!!
"hello" == "world" # true !!! Don't compare strings as if they were numbers!
"hello" eq "world" # false
"hello" == ""    # true !!! Don't compare strings as if they were numbers!
"hello" eq ""    # false

```

### Example 5-12. examples/scalars/comparison\_operators.p6

```
#!/usr/bin/perl6
```

```

use v6;

say 4      == 4 ?? "TRUE" !! "FALSE";    # TRUE
say 3      == 4 ?? "TRUE" !! "FALSE";    # FALSE
say "3.0"  == 3 ?? "TRUE" !! "FALSE";    # TRUE
say "3.0"  eq 3 ?? "TRUE" !! "FALSE";    # FALSE
say 13     > 2 ?? "TRUE" !! "FALSE";    # TRUE
say 13     gt 2 ?? "TRUE" !! "FALSE";    # FALSE
say "foo"  == "" ?? "TRUE" !! "FALSE";   # TRUE
say "foo"  eq "" ?? "TRUE" !! "FALSE";   # FALSE
say "foo"  == "bar" ?? "TRUE" !! "FALSE"; # TRUE
say "foo"  eq "bar" ?? "TRUE" !! "FALSE"; # FALSE

```

## 5.13. Boolean expressions (logical operators)

**Table 5-2. Logical operators**

and	&&
or	
orelse	//
xor	^^
not	!

```

if COND and COND {
}

```

```

if COND or COND {
}

```

```

if not COND {
}

```

### Example 5-13. examples/scalars/logical\_operators.p6

```

#!/usr/bin/perl6
use v6;

say (2 and 1); # 1
say (1 and 2); # 2
say (1 and 0); # 0
say (0 and 1); # 0
say (0 and 0); # 0
say "---";

say (1 or 0); # 1
say (1 or 2); # 1
say (0 or 1); # 1
say (0 or 0); # 0

```

```

say "---";

say (1 // 0);      # 1
say (0 // 1);      # 0
say (0 // 0);      # 0
say (undef // 0); # 0
say (undef // 1); # 1
say "---";

say (1 xor 0);     # 1
say (0 xor 1);     # 1
say (0 xor 0);     # 0 TODO??
say (1 xor 1);     #
say "---";

say (not 1);       # 0 TODO ?
say (not 0);       # 1
say (not undef);  # 1
say "---";

```

## 5.14. Chained comparisons

### Example 5-14. examples/scalars/chained\_comparison.p6

```

#!/usr/bin/perl6
use v6;

my $a = prompt "Type in a number between 23 and 42: ";
if (23 <= $a and $a <= 42) {
    say "Good, $a is in the range.";
} else {
    say "Did I say between 23 and 42 ?";
}

# you can also compare like this
if (23 <= $a <= 42) {
    say "Good, $a is in the range.";
} else {
    say "Did I say between 23 and 42 ?";
}

my $small = prompt "Type another number between 0 and $a: ";
my $big = prompt "Type another number between $a and 100: ";

if (0 <= $small <= $a <= $big <= 100) {
    say "good";
}

```

```

} else {
    say "something is fishy";
}

```

## 5.15. Comparing values - Calculator

### Example 5-15. examples/scalars/calculator.p6

```

#!/usr/bin/perl6
use v6;

my $a          = prompt "Number:";
my $operator   = prompt "Operator: [+-*/]:";
my $b          = prompt "Number:";

if $operator eq "+" {
    say $a + $b;
} elsif $operator eq "-" {
    say $a - $b;
} elsif $operator eq "*" {
    say $a * $b;
} elsif $operator eq "/" {
    say $a / $b;
} else {
    say "Invalid operator $operator";
}

```

## 5.16. Calculator - given

The *given - when* construct (known in other languages as case or switch) can make the previous example much more compact.

Perl compares the value of `$operator` (the topic) with each one of the values next to the `when` keywords. When it finds one that fits the appropriate block is executed and perl jumps to the next command after the `given` block.

If none of the `when` values fit the (optional) default block is evaluated.

### Example 5-16. examples/scalars/calculator\_given.p6

```

#!/usr/bin/perl6
use v6;

my $a          = prompt "Number:";
my $operator   = prompt "Operator: [+-*/]:";

```

```

my $b          = prompt "Number:";

given $operator {
    when "+" { say $a + $b; }
    when "-" { say $a - $b; }
    when "*" { say $a * $b; }
    when "/" { say $a / $b; }
    default { say "Invalid operator $operator"; }
}

```

## 5.17. String functions: index

### Example 5-17. examples/scalars/string\_functions\_index.p6

```

#!/usr/bin/perl6
use v6;

my $s = "The black cat jumped from the green tree";

say index $s, "e";           # 2
say index $s, "e", 3;      # 18

say rindex $s, "e";        # 39
say rindex $s, "e", 38;   # 38
say rindex $s, "e", 37;   # 33

```

## 5.18. String functions: substr

### Example 5-18. examples/scalars/string\_functions\_substr.p6

```

#!/usr/bin/perl6
use v6;

my $s = "The black cat climbed the green tree";
my $z;
$z = substr $s, 4, 5;           # $z = black
say $z;
$z = substr $s, 4, -11;       # $z = black cat climbed the
say $z;
$z = substr $s, 14;           # $z = climbed the green tree
say $z;
$z = substr $s, -4;           # $z = tree
say $z;
$z = substr $s, -4, 2;        # $z = tr
say $z;

```

## 5.19. Super-or (junctions)

### Example 5-19. examples/scalars/junctions.p6

```
#!/usr/bin/perl6
use v6;

say "Please select an option:";
say "1) one";
say "2) two";
say "3) three";
my $c = prompt("");

if ($c == 1 or $c == 2 or $c == 3) {
    say "correct choice: $c";
} else {
    say "Incorrect choice: $c";
}

if ($c == 1|2|3) {
    say "correct choice: $c";
} else {
    say "Incorrect choice: $c";
}
```

# Chapter 6. Files in Perl6

## 6.1. exit, warn, die

### Example 6-1. examples/files/exit.p6

```
#!/usr/bin/perl6
use v6;

say "hello";

exit;

say "world";
```

### Example 6-2. examples/files/warn\_die.p6

```
#!/usr/bin/perl6
use v6;

warn "This is a warning";

say "Hello World";

# TODO: Parrot still gives a stack trace when calling die..
#die "This will kill the script";

#say "This will not show up";
```

## 6.2. Read line from file

As in other high level languages one has to open a file in order to read from it or to write to it. In Perl6 it is done by the *open* function that receives two parameters. The name of the file and the mode. In order to open a file for reading the mode need to be *:r*. the function either returns a file handle that should be placed in a scalar variable or returns undef.

```
$fh = open $filename, :r
```

Once we have an open file handler we can use the `$fh.readline` or the `=>$fh` syntax to read one line (the next line) from the given file.

**Example 6-3. examples/files/read\_one\_line.p6**

```
#!/usr/bin/perl6
use v6;

my $filename = "examples/files/read_one_line.p6";

if (my $fh = open $filename, :r) {
    my $line = $fh.readline;
    say $line;
} else {
    say "Could not open '$filename'";
}
```

## 6.3. Process a file line by line

**Example 6-4. examples/files/read\_file.p6**

```
#!/usr/bin/perl6
use v6;

my $filename = "examples/files/read_file.p6";

if (my $fh = open $filename, :r) {
    for $fh.readline -> $line {
        say $line;
    }
} else {
    say "Could not open '$filename'";
}
```

The *for* loop in our example repeatedly calls of `$fh.readline` (or `= $fh`) and on each iteration it places the retrieved string in the `$line` variable.

This script is very similar to what the unix `cat` command does.

## 6.4. Write to a file

**Example 6-5. examples/files/write\_file.p6**

```
#!/usr/bin/perl6
use v6;

my $filename = "temp.txt";

if (my $fh = open $filename, :w) {
```

```

    $fh.print("Your name: ");
    my $name = =*$IN;
    $fh.say($name);
}

```

## 6.5. Open file modes

```

:r - read
:w - write
:a - append

```

## 6.6. Exercise: Print sum of numbers

Take a file that has one number on every line  
Print the sum of the numbers

Improve it and also print average, minimum and maximum.

Can you also think about way to print median and standard deviation?

**Example 6-6. examples/numbers.txt**

```

3
8
19
-7
13

```

## 6.7. Solution

**Example 6-7. examples/files/statistics.p6**

```

#!/usr/bin/perl6
use v6;

my $filename = 'examples/numbers.txt';

my $total;
my $count;
my $min;
my $max;

if (my $fh = open $filename, :r) {

```

```

for $fh.readline -> $line {
    if (not $count) {
        $min = $max = $line;
    }
    $total += $line;
    if ($min > $line) {
        $min = $line;
    }
    if ($max < $line) {
        $max = $line;
    }
    $count++;
}
my $average = $total / $count;
say "Total: $total, Count: $count Average: $average Min: $min Max: $max";
} else {
    say "Could not open '$filename'";
}

# There is a minor issue in this solution, what if there are no values at all in the file?

```

## 6.8. undef, defined, not

If you declare a variable using `my` but don't give it a value it still has a special value called *undef*.

By using the *defined* keyword one can tell if a scalar value has any real value or it is `undef`.

As we will see later this `undef` value is also used to indicate some kind of a failure.

The usage would look like this:

```

my $x;
if (defined $x) {
    say "It has a value";
}

```

Sometimes the opposite case is what needs some action so we will use the *not* operator to negate the trueness of the statement:

```

if (not defined $x) {
    say "Do something as x is not yet defined";
}

```

# Chapter 7. Perl6 Lists and Arrays

## 7.1. List Literals, list ranges

Things in () separated by commas are called a list of things.

A list is an ordered set of scalar values.

Examples of lists:

### Example 7-1. examples/arrays/list\_literals.p6

```
#!/usr/bin/perl6
use v6;

(1, 5.2, "apple");           # 3 values

(1,2,3,4,5,6,7,8,9,10);     # nice but we are too lazy, so we write this:
(1..10);                    # same as (1,2,3,4,5,6,7,8,9,10)
(1..Inf);                   # represent the list of all the numbers
(1..*);                     # this too

("apple", "banana", "peach", "blueberry"); # is the same as
<apple banana peach blueberry>;          # quote word

my ($x, $y, $z);           # We can also use scalar variables as elements of a list
```

## 7.2. List Assignment

```
my ($x, $y, $z);
($x, $y, $z) = f(); # if f() returns (2, 3, 7) then it is nearly the same as $x=2; $y=3; $z=7
($x, $y, $z) = f(); # if f() returns (2, 3, 7, 9), then ignore 9
($x, $y, $z) = f(); # if f() returns (2, 3); then $z will be undef
```

A regular question on job interviews:

How can we swap the values of 2 variables, let say \$x and \$y?

## 7.3. Swap two values

### Example 7-2. examples/arrays/lists.p6

```
#!/usr/bin/perl6
use v6;
```

```

say "Type in two values:";
my $a = =$*IN;
my $b = =$*IN;

($a, $b) = ($b, $a);
say $a;
say $b;

```

## 7.4. loop over elements of list with for

### Example 7-3. examples/arrays/arrays.p6

```

#!/usr/bin/perl6
use v6;

for "Foo", "Bar", "Baz" -> $name {
    say $name;
}

say "---";

for 1..5 -> $i {
    say $i;
}

say "---";

# TODO
#for 1..Inf -> $i {
#    say $i;
#    last if $i > 10;
#}
#
#say "----";

```

## 7.5. Create array, go over elements

Arrays start with a "@" mark and the name of the array.  
You can assign a list of values to the array.

Printing the array as is, shows them with no space between them.  
One can also put the array within a string (interpolating) but then it needs to be enclosed in {} curly braces. This will print spaces between the values.

The for loop lets you iterate through the values of the array.

**Example 7-4. examples/arrays/list\_colors\_array.p6**

```
#!/usr/bin/perl6
use v6;

my @colors = ("Blue", "Yellow", "Brown", "White");
say @colors;

say "-----";           # just for separation...

say "{@colors}";

say "-----";           # just for separation...

for @colors -> $color {
    say $color;
}
```

Output:

```
BlueYellowBrownWhite
-----
Blue Yellow Brown White
-----
Blue
Yellow
Brown
White
```

## 7.6. Array elements (create menu)

**Example 7-5. examples/arrays/color\_menu.p6**

```
#!/usr/bin/perl6
use v6;

my @colors = <Blue Yellow Brown White>;

for 1..@colors.elems -> $i {
    say "$i) {@colors[$i-1]}";
}

print "Please select a number: ";
my $input = =$*IN;
say "You selected {@colors[$input-1]}";
```

## 7.7. Array assignment

### Example 7-6. examples/arrays/array\_assignment.p6

```
#!/usr/bin/perl6
use v6;

my $owner = "Moose";
my @tenants = "Foo", "Bar";
my @people = ($owner, 'Baz', @tenants); # the array is flattened:
say "@people";                          # Moose Baz Foo Bar

my ($x, @y) = (1, 2, 3, 4);
say $x;                                  # $x = 1
say "@y";                                # @y = (2, 3, 4)
```

## 7.8. Command line options

### Example 7-7. examples/arrays/color\_menu\_option.p6

```
#!/usr/bin/perl6
use v6;

my $color = @*ARGS[0];

if (not $color) {
    my @colors = ("Blue", "Yellow", "Brown", "White");

    for 1..@colors.elems -> $i {
        say "$i) {@colors[$i-1]}";
    }

    print "Please select a number: ";
    my $input = =*$IN;
    $color = @colors[$input-1];
}

say "You selected $color";
```

## 7.9. Process CSV file

### Example 7-8. examples/arrays/process\_csv\_file.csv

```
Foo,Bar,10,home
Orgo,Morgo,7,away
```

```
Big, Shrek, 100, US
Small, Fiona, 9, tower
```

### Example 7-9. examples/arrays/process\_csv\_file.p6

```
#!/usr/bin/perl6
use v6;

my $file = 'examples/arrays/process_csv_file.csv';
if (defined @*ARGS[0]) {
    $file = @*ARGS[0];
}

my $sum = 0;
my $data = open($file, :r);
for =$data -> $line {
    my @columns = split ",", $line;
    $sum += @columns[2];
}
say $sum;
```

## 7.10. join

### Example 7-10. examples/arrays/join.p6

```
#!/usr/bin/perl6
use v6;

my @fields = <Foo Bar foo@bar.com>;
my $line = join ";", @fields;
say $line;      # Foo;Bar;foo@bar.com
```

## 7.11. Hyper Operators

### Example 7-11. examples/arrays/hyper.p6

```
#!/usr/bin/perl6
use v6;

my @a = (1, 2, 3);
my @b = (7, 10, 12);

my @c = @a &+& @b;
say join ", ", @c;

#TODO
```

```

#my @d = @c >>+<< 3;  ##???
#say "@d";

# my @e = @c >>+>> 3;  ##???

#my @x = (1, 2, 3) >>+<< (4, 5);  # 5 7 3
#say "@x";

```

## 7.12. slurp

### Example 7-12. examples/files/slurp\_file.p6

```

#!/usr/bin/perl6
use v6;

my $filename = 'examples/files/slurp_file.p6';

my $data = slurp $filename;
say $data.bytes;

my @rows = slurp $filename;
say @rows.elems;

```

## 7.13. xx - string multiplier

### Example 7-13. examples/arrays/xx.p6

```

#!/usr/bin/perl6
use v6;

my @moose = "moose" xx 3;
say "@moose";

```

# Chapter 8. Perl6 Hashes

## 8.1. Associative Arrays (Hashes)

A hash (also called associative array) is a set of key,value pairs where the keys are unique strings and the values can have any, err value.

Hashes always start with a % (percentage) sign.

### Example 8-1. examples/hash/create\_hash.p6

```
#!/usr/bin/perl6
use v6;

my %user_a = ("fname", "Foo", "lname", "Bar");

my %user_b = (
    "fname" => "Foo",
    "lname" => "Bar",
);

say %user_a{"fname"};
%user_a{"email"} = "foo@bar.com";
say %user_a{"email"};

say %user_b<lname>;
```

## 8.2. Fetching data from a hash

### Example 8-2. examples/hash/print\_hash.p6

```
#!/usr/bin/perl6
use v6;

my %user = (
    "fname" => "Foo",
    "lname" => "Bar",
    "email" => "foo@bar.com",
);

for %user.keys.sort -> $key {
    say "$key {%user{$key}}";
}
```

## 8.3. Multidimensional hashes

Will be supported but AFAIK not yet implemented in Pugs.  
 You might want to help adding them: <http://www.pugscode.org/>

## 8.4. Count words

### Example 8-3. examples/hash/count\_words.p6

```
#!/usr/bin/perl6
use v6;

my $filename = 'examples/words.txt';

my %counter;

my $fh = open($filename, :r) // die "Could not open '$filename'";
for =$fh -> $line {
    my @words = split /\s+/, $line;
    for @words -> $word {
        %counter{$word}++;
    }
}

for %counter.keys.sort -> $word {
    say "$word {%counter{$word}}";
}
```

## 8.5. Overview

### Example 8-4. examples/hash/hash.p6

```
#!/usr/bin/perl6
use v6;

# creating hashes
my %h1 = (first => '1st', second => '2nd');

if (%h1{'first'}.defined) {
    say "the value of 'first' is defined";
}
if (%h1<second>.defined) {
    say "the value of 'second' is defined";
}

if (%h1.exists('first')) {
    say "the key 'first' exists in h2";
}
```

```

}

say %h1.exists('third') ?? "third exists" !! "third does not exist";

say %h1<first>;
say %h1<second>;

# TODO hash with fixed keys not implemented yet
#my %h2{'a', 'b'} = ('A', 'B');
#say %h2.delete('a');
#say %h2.delete('a');

```

## 8.6. slurp

### Example 8-5. examples/files/slurp\_csv\_file.p6

```

#!/usr/bin/perl6
use v6;

my $filename = 'examples/phonebook.txt';

my @lines = slurp $filename;
for @lines -> $line {
    print "L: $line";
}

my %phonebook = map {split ",", $_[0]}, @lines;

print "Name:";
my $name = =*$IN;
say %phonebook{$name};

```

### Example 8-6. examples/phonebook.txt

```

Foo,123
Bar,78
Baz,12321

```

## 8.7. kv

### Example 8-7. examples/hash/print\_hash\_kv.p6

```

#!/usr/bin/perl6
use v6;

my %user = (
    "fname" => "Foo",

```

```
    "lname" => "Bar",  
    "email" => "foo@bar.com",  
);  
  
for %user.kv -> $key, $value {  
    say "$key $value";  
}
```

# Chapter 9. Subroutines in Perl6

For more details look at S06.

## 9.1. Simple definition with required parameters

\* *In order to define a subroutine in Perl 6 we use the "sub" keyword. Then comes the name of the subroutine with a list of parameters. After that comes a block that implements the subroutine where you can use the variables defined in the signature. In the most simple case shown in the example we have two scalar variables (\$a and \$b) as parameters. In this case the parameters are required. The user of the sub has to supply them though there are several ways to do that. The simple way is just to provide them in the same order as they are shown in the signature.*

Positional parameters:

### Example 9-1. examples/subroutines/required\_params.p6

```
#!/usr/bin/perl6
use v6;

sub add ($a, $b) {
    return $a + $b;
}

say add(2, 3);      # returns 5

#say add(2);      # is an error
#say add(2, 3, 4); # is an error
```

\* *The user can decide to pass the parameters in as key-value pairs.*

Named parameters:

### Example 9-2. examples/subroutines/required\_params\_named.p6

```
#!/usr/bin/perl6
use v6;

sub add (:$a, $b) {
    return $a + $b;
}

say add(b => 2, a => 3);      # returns 5

#say add(2);      # is an error
#say add(2, 3, 4); # is an error
```

## 9.2. Named parameters

### Example 9-3. examples/subroutines/named\_definitions.p6

```
#!/usr/bin/perl6
use v6;

sub add ($a, $b) {
    return $a + $b;
}

say "first: ", add(2, 3);      # returns 5
#say add(2);                 # is an error
#say add(2, 3, 4);          # is an error

sub sum (@numbers) {
    my $sum = 0;
    for @numbers -> $num {
        $sum += $num;
    }
    return $sum;
}

# Now you can already call sum(2) and sum(2, 3) and sum(2, 3, 4) and they
say sum(2); #??
say sum((2, 3));
say sum((2, 3, 4));

sub sumsum (*@numbers) {
    my $sum = 0;
    for @numbers -> $num {
        $sum += $num;
    }
    return $sum;
}

say sumsum(2);
say sumsum(2, 3);
say sumsum(2, 3, 4);
say sumsum((2, 3, 4));
```

## 9.3. Subroutines

### Example 9-4. examples/subroutines/optional\_params.p6

```
#!/usr/bin/perl6
use v6;

# search the text within a file return 1 if found, 0 if not
```

```

sub search ($text, $file) {
    my $fh = open $file, :r orelse die;
    for $fh.readline -> $line {
        if index($line, $text) > -1 {
            return 1;
        }
    }
    return 0;
}

# optional parameter
sub search($text, $file, $all?) {
    my $fh = open $file, :r orelse die;
    my $cnt = 0;
    for $fh.readline -> $line {
        if index($line, $text) > -1 {
            return 1 if not $all;
            $cnt++;
        }
    }
    return $cnt;
}

```

## 9.4. No parameter definition - perl 5 style

\* You can also use the old Perl 5 style way to defined functions. In such case you would not define a signature and any value passed to the subroutine will show up in the internal `@_` variable. Any array or hash will be flattened.

### Example 9-5. examples/subroutines/definitions.p6

```

#!/usr/bin/perl6
use v6;

my $z = add(19, 23);
say "first ", $z;

say "second ", add2(19, 23);

say "third ", add3(2, 3);

sub add {
    my ($x, $y) = @_;
    return $x + $y;
}

sub add2 ($x, $y) {
    return $x + $y;
}

sub add3 (Int $x, $y) {

```

```

    return $x + $y;
}

```

## 9.5. Subroutines

### Example 9-6. examples/subroutines/fibonacci.p6

```

#!/usr/bin/perl6
use v6;

my $N = 10; # @*ARGS;

say "Computing Fibonacci of {$N}";
say "Result: " ~ fib_recursive($N);
say "Result: " ~ fib_r($N);
say "All: " ~ join " ", fib($N);

sub fib_recursive ($n) {
    return 1 if $n == 1 or $n == 0;
    return fib_recursive($n-1)+fib_recursive($n-2);
}

sub fib_r ($n) {
    return 1 if $n == (1|0);
    return fib_r($n-1)+fib_r($n-2);
}

sub fib ($n) {
    my @fibs;

    @fibs.push(1) if $n > 0;
    @fibs.push(1) if $n > 1;

    for 2..$n {
        @fibs.push(@fibs[*-1]+@fibs[*-2]);
    }
    return @fibs;
}

```

## 9.6. Subroutines

### Example 9-7. examples/subroutines/multiple\_same\_name\_params.p6

```

#!/usr/bin/perl6
use v6;

```

```
# TODO named parameters
f(1, 4);           # 1 4
#f(y => 6, x => 2); # 2 6
#f(2, y => 6);     # 2 6
#f(2, x => 6);     # 6 2

sub f($x, $y) {
    say "$x $y";
}

sub u($x, $y?) {
}

sub x($x, $y = 7) {
}
```

# Chapter 10. Perl 6 Regexes

## 10.1. Regexes in Perl 6

Let's start by saying that in Perl 6 we call it a Regex not Regular Expression but you'd be forgiven if you used either name. Just don't get into an argument with some from an Univeristy.

## 10.2. Simple matching

In Perl 6 the smart match `~~` operator is used for regex matching.

For negative matching use the `!~~` operator.

### Example 10-1. examples/regex/simple.p6

```
#!/usr/bin/perl6
use v6;

my $str = 'abc123';

if ($str ~~ /a/) {
    say "Matching";
}

if ($str !~~ /z/) {
    say "No z in $str";
}
```

\* Any non-alphanumeric character needs to be escaped, even if they don't currently have any special meaning or you'll get a "Statement not terminated properly" error during compilation.

### Example 10-2. examples/regex/special\_chars.p6

```
#!/usr/bin/perl6
use v6;

my $str = 'abc = 123';

if ($str ~~ /\=/) {
```

```
    say "Matching";  
}
```

## 10.3. Substitute

### Example 10-3. examples/regex/substitute.p6

```
#!/usr/bin/perl6  
use v6;  
  
my $str = 'abc = 123';  
  
$str ~~ s/\=/x/;  
say $str;
```

# Chapter 11. Junctions in Perl6

## 11.1. Junctions

As we already seen in a previous chapter we can use a junction to check if a values is among a given set of values

### Example 11-1. examples/junctions/intro.p6

```
#!/usr/bin/perl6
use v6;

my $possible_options = 1|2|3;
print "please give a number(1,2 or 3): ";
my $c = =$*IN;
if ($c == $possible_options) {
    say "Correct choice: $c";
} else {
    say "Incorrect choice: $c";
}
```

## 11.2. More examples with Junctions

### Example 11-2. examples/junctions/operators.p6

```
#!/usr/bin/perl6
use v6;

my $options = 1|2|3;

$options *= 2;    # now it has 2|4|6

for 1..8 -> $i {
    if ($i == $options) {
        say $i;
    }
}
```

### Example 11-3. examples/junctions/substr.p6

```
#!/usr/bin/perl6
use v6;

my $x = 0|6;
```

```

my $str = "Hello World";
my $sub = substr($str, $x, 5);

my @values = $sub.values;
say "{@values}";          # Hello World
say $sub.values[0];      # Hello
say $sub.values.elems;   # 2 - the number of elements in the junction

```

**Example 11-4. examples/junctions/j.p6**

```

#!/usr/bin/perl6
use v6;

my $x = 1|2;                                     # junction
if ($x == 1) { say 't' } else { say 'f' } # t
if ($x == 2) { say 't' } else { say 'f' } # t
if ($x == 3) { say 't' } else { say 'f' } # f
if ($x != 1) { say 't' } else { say 'f' } # t

#TODO
#my @values = $x.values;                         # fetch the values of a junction, order is random
#say $x.values.elems;                            # 2
#say $x.values[0] + $x.values[1]; # 3
#
#my $y = 1|1|2;
#say $y.values.elems;                            # 2
#say $y.values[0] + $y.values[1]; # 3
## 1|1|2 is the same as 1|2
#
#my $r1 = (1|2 == 1); # (Bool::False | Bool::True)
#my $r2 = (1|2 == 2); # (Bool::False | Bool::True)
#
#my $r3 = (1|2 == 3); # (Bool::False)
#
#           # there is only one element because multiple same booleans are irrelevant
#           # this is the same reduction we saw with 1|1|2
#

```

**Example 11-5. examples/junctions/x.p6**

```

#!/usr/bin/perl6
use v6;

#my $x = any(1,2,3);
#my $z = any(1);
#say $x - $z;

my @numbers = (1, 2, 3);
my @new = (5, 3);
if (all(@new) > all(@numbers)) {
    say "all bigger";
}

```

```

if (any(@new) > all(@numbers)) {
    say "there is at least one bigger";
}

#my $diff = all(@numbers) and none(@new);
#say $diff.perl;

```

**Example 11-6. examples/junctions/any.p6**

```

#!/usr/bin/perl6
use v6;

my @names = <Foo Bar Moo>;
print "Please type in your name: ";
my $username = =$*IN;
if $username eq any(@names) {
    say "Welcome $username";
} else {
    say "Unknown user $username";
}

#if ($username eq any(@names)) {
#    say "Welcome $username";
#}

```

# Chapter 12. Perl5 to Perl6

## 12.1. Intro

While there are many similarities between Perl5 and Perl6 there are also a few substantial changes. This chapter tries to ease the transition for Perl5 programmers to Perl6.

In this chapter you won't find strongest operators of Perl6. What I am trying to show here is that given a Perl5 program how one can more-or-less straight forward translate it to Perl6.

See also the documentation <http://svn.pugscode.org/pugs/docs/Perl6/>

## 12.2. Hello World

### Example 12-1. examples/p526/hello\_world.p6

```
#!/usr/bin/perl6
use v6;

# Perl5: print
print "Hello World\n"; # still works
say "Hello World";     # the same as print but adds a \n automatically
```

## 12.3. Scalars

### Example 12-2. examples/p526/scalars.p6

```
#!/usr/bin/perl6
use v6;

# Perl5: my
my $name = "Moose"; # you always need "my"
say "Hello $name";  # still interpolates

# Perl5: <STDIN>
my $line = =*$IN;   # reading from standard input, chomps automatically

# Perl5: substr
say substr $line, 2, 3; # substr works as before (also the 4 paramter version)
say $line.substr(2, 3); # also works this way
```

```

# Perl5: length
my $a = "This is a string";
say chars $a;           # the number of characters (was length)
say $a.chars;         # also works

#say bytes $a;         # the number of bytes
#say $a.bytes;        # also works

# Perl5: chomp
my $b = chomp $a;     # returns the chomped string, does NOT ! change $a;
my $c = $a.chomp;    # also works

# Perl5: defined
defined $b;           # check if $b is defined or "undef"
$b.defined;          # also works

# Perl5: undef
$b = undef;          # set $b to "undef"

# Perl5: qq
# Perl5: q

# Perl5: local
# temp

# Perl5: $$, $@, $!, ...
# Perl5: $_
#
# Perl5: $0
# $?FILE

# Perl5: %ENV
# %*ENV

```

## 12.4. Arrays

### Example 12-3. examples/p526/arrays.p6

```

#!/usr/bin/perl6
use v6;

my @perl = ("one", "two", "three"); # creating an array
# Perl5: qw
my @perl6 = <one two three>;      # ceating an array

```

```

say @perl;                # onetwothree      just as in the old days
say "@perl";             # @a          it does not interpolate any more
say "joe@perl.org";     # joe@perl.org so this works now

# Perl5: array interpolation
say "{@perl}";          # one two three  is the way to interpoalte an array

# Perl5: array element
say @perl[0];           # one           when accessing the elements of an a

# Perl5: number of elements in array
say elems(@perl);      # 3            number of elements
say @perl.elems;      # also works

```

## 12.5. Hashes

### Example 12-4. examples/p526/hashe.p6

```

#!/usr/bin/perl6
use v6;

my %h = ("Perl", "Larry", "Python", "Guido"); # creates hash as in Perl5
# Perl5: hash element
say %h{"Perl"}; # accessing hash elements is simialr but
# Perl5: hash key without quotes
say %h<Perl>; # if you want to leave out the quotation

# Perl5: exists
%h.exists("Perl"); # checks if key exists (formally the exi
defined(%h<Perl>); # checks if the value of the give key is

# Perl5: keys
keys %h; # returns a list of keys just as in Perl
%h.keys; # also works

# Perl5: values

# Perl5: each
# %h.kv
for %h.kv -> $key, $value {
    say "$key => $value";
}

```

## 12.6. Control Structures

### Example 12-5. examples/p526/control\_structures.p6

```
#!/usr/bin/perl6
use v6;

# Perl5: if
# No need to for () around the condition
my $x = 23;
if $x < 42 {
}

# Perl5: foreach (for)
my @names = <Larry Guido Matz>;
for @names -> $person {
    say $person;
}

# Perl5: for (foreach)
loop (my $i=1; $i < 10; $i++) {
    say $i;
}

# infinite loop:
loop {
    # well, almost infinite :-)
    last;
}

# Perl5: while
while $i < 10 {
    say $i;
}
```

## 12.7. Functions

### Example 12-6. examples/p526/functions.p6

```
#!/usr/bin/perl6
use v6;

my $lang = "Perl";

creator("Larry", $lang);
sub creator {
    my ($person, $language) = @_;
    # still works as in Perl5 and you get the value
    # you can copy them
```

```

    say "the creator of $language is $person";
}

creator_2("Larry", $lang);
sub creator_2 ($person, $language) {
    # $language ~= 6;
    say "the creator of $language is $person";
}

creator_3("Larry", $lang);
sub creator_3 ($person, $language is rw) {
    $language ~= 6;
    say "the creator of $language is $person";
}
say "What ? $lang";

creator_4("Larry", $lang);
sub creator_4 ($person is copy, $language is rw) {
    $person ~= " Wall";
    say "the creator of $language is $person";
}

```

## 12.8. Files

### Example 12-7. examples/p526/files.p6

```

#!/usr/bin/perl6
use v6;

# Perl5: open
my $fh = open 'examples/p526/files.p6', :r;

# Perl5: close
close($fh);
$fh.close

# Perl5: <$fh>
# $fh.readline

```

## 12.9. Modules, Classes

### Example 12-8. examples/p526/modules.p6

```
#!/usr/bin/perl6
use v6;

# Perl5: package
module Person {
    our $fname;
    our $lname;
}
$Person::fname = "Foo";
$Person::lname = "Bar";
```

## 12.10. Perl 5 to Perl6

```
@ARGV          @$ARGS
$0             $PROGRAM_NAME
foreach my $s (@arr) {} for @arr -> $s { }
```

```
$array
scalar @array   @array.elems  (@array.length not yet implemented ?)
```

# Chapter 13. Shell to Perl6

## 13.1. Intro

Perl5 already was a good replacement for any shell programming.  
With the new features Perl6 improves this even further.

In this section we will see several examples especially for shell programmers.

## 13.2. Unix commands in Perl6

In shell script we don't usually read in a file to memory, instead we do several passes on the same file in order to extract data.

In a high-level language such as Perl6 we can read in all the content of the file and have better ways to fetch various pieces of information. This ease is of course only relevant for files that can fit into memory. If we have to deal with files larger than the free memory in the computer we cannot read all of it into memory so either we have to do several passes on a file or make our algorithm more clever.

## 13.3. awk

This needs more than a one page explanation I think.

## 13.4. cat

In shell scripts cat is usually used to read in a file to be processed by other tools  
in Perl6 you can slurp in a file:

```
my @rows = "data.txt".slurp;  
my @rows = slurp "data.txt";
```

but you can also read them as one sting:

```
my $file = slurp "data.txt";
```

It is also used to its original purpose - to concatenate several files -  
UNIX: cat a.txt b.txt > new.txt

```
Perl6:  
my $out = open "new.txt", :w err die "Could not open new.txt";  
$out.say("a.txt".slurp)
```

## 13.5. cd in Perl6

chdir - changing the current working directory

## 13.6. chmod

chmod - changing the user rights on a file or a directory

## 13.7. chown

chown - change the ownership of the file  
not yet implemented in Pugs

## 13.8. cmp

Probably will have an external module implementing its behavior partially.

Perl5: there is limited support to this in File::Compare

## 13.9. compress

Probably in external module.

## 13.10. cut

## 13.11. date

Perl5: POSIX::strftime

## 13.12. diff

## 13.13. df

## 13.14. dos2unix

## 13.15. du

## 13.16. file

No implementation in Perl5 or 6

```
my $result = 'file $filename';
```

## 13.17. find

The File::Find standard module provides comparable functionality.

## 13.18. grep

The grep command takes a pattern and for each input row it decides if the pattern matches on that row.

The grep function can also take a pattern and for each element in the input list decide if the pattern matches.

```
grep PATTERN FILE
```

```
"FILE".slurp.grep(/PATTERN/).
```

## **13.19. gzip**

## **13.20. head**

## **13.21. kill**

## **13.22. ln**

## **13.23. ls**

## **13.24. mkdir**

## **13.25. mv**

## **13.26. ps**

## 13.27. popd

## 13.28. pushd

## 13.29. pwd

## 13.30. rmdir

## 13.31. rm

## 13.32. sed

## 13.33. sort

Perl has an internal sort function that can even sort on arbitrary criteria.

```
"input.txt".slurp.sort.say          will sort the rows of the line according to the ASCII table  
"input.txt".slurp.sort:{ $^a <=> $^b }.say    will sort the rows
```

## 13.34. tail

tail displays the last few lines of a file or from a listing received via pipe.  
It is often used to see the last few entries in a file listing.

There is no direct implementation in Perl6 but one can easily read.

## 13.35. tar

There is no implementation in Perl6.

In Perl 5 one can use the Archive::Tar module for this.

## 13.36. touch

Is not directly implemented. Most of its features can be easily implemented.

```
if (my $f = open("filename", :r)) { $f.close}; # is touch -a
if (my $f = open("filename", :a)) { $f.close}; # is touch
if (my $f = open("filename", :w)) { $f.close}; # will create the file if it does not exist
```

## 13.37. uniq

The UNIX `uniq` command drops the duplicate values from its input and prints out the result. Usually it is used in a pipe so it receives file rows and prints them on the screen (or passes them to the next command in the pipe chain).

In Perl6 we have a `uniq` function that receives a list of values and returns the same list after dropping any duplicate value.

```
"input.txt".slurp.uniq.map: { "$_\n" }.say
```

and this will work once `slurp` is fixed:

```
"input.txt".slurp.uniq.say
```

will print out the `uniq` rows in the `input.txt` file to the screen

## 13.38. unix2dos

## 13.39. wc

## 13.40. who

## 13.41. zip

## 13.42. Other UNIX command

setenv/set/alias  
processing all the files in a directory (or in a tree)

which  
bc

bash: for i in \*; do echo \$i; done

# Chapter 14. Object Oriented Perl6

## 14.1. Simple Point class

\* In Perl 6 we define classes with the "class" keyword. It can be used either as the first line of a file or in front of a block defining the class. The has keyword is used to create attributes and the various twigils can be used to fine tune the meaning of those attributes. Using dot (.) as the twigil will create a public attribute meaning it will automatically create and accessor for that attribute. When a class is defined it automatically provides a constructor called new that can get a hash to set the attributes. WHAT helps with introspection so we can ask what class does an object belong to. By default the accessors are read only so you cannot assign to them.

### Example 14-1. examples/classes/point\_01.p6

```
use v6;

class Point {
  has $.x;
  has $.y;
};

my $a = Point.new(x => 23, y => 42);

say $a.WHAT;      # Point

say $a.x;        # 23

#$a.x = 10;      # Exception: Cannot assign to readonly variable.
```

## 14.2. Read/write attributes and accessors

\* Setting the "rw" trait on the attribute will also generate a read-write accessor that is an lvalue so you assign to it.

### Example 14-2. examples/classes/point\_02.p6

```
use v6;

class Point {
  has $.x is rw;
  has $.y is rw;
};

my $a = Point.new(x => 23, y => 42);
```

```
say $a.x;    # 23

$a.x = 10;

say $a.x;    # 10
```

## 14.3. Class Methods

\* The "method" keyword can be used to create methods for the class.

### Example 14-3. examples/classes/point\_03.p6

```
use v6;

class Point {
  has $.x is rw;
  has $.y is rw;

  method reset {
    $.x = 0;
    $.y = 0;
  }
};

my $a = Point.new(x => 23, y => 42);

say $a.x;    # 23

$a.reset;

say $a.x;    # 0
```

## 14.4. Private Attributes

\* Using exclamation mark as the twigil indicates that the attribute is private. So has \$!x will create a variable that can be accessed from within the class but which won't have an accessor from the outside world.

### Example 14-4. examples/classes/point\_04.p6

```
use v6;

class Point {
  has $.x is rw;
  has $.y is rw;

  has $!weight;

  method reset {
```

```

    $.x = 0;
    $.y = 0;
    $!weight = 0;
  }
};

my $a = Point.new(x => 23, y => 42, weight => 2);

say $a.x; # 23
#say $a.weight; # Exception: Could not locate a method 'weight' to invoke on class 'Point'

$a.reset;

say $a.x; # 0

```

## 14.5. Method with parameters

\* *Methods can have parameters, just as any subroutine would have.*

### Example 14-5. examples/classes/point\_05.p6

```

use v6;

class Point {
  has $.x is rw;
  has $.y is rw;

  method reset {
    $.x = 0;
    $.y = 0;
  }

  method set_coordinates($x, $y) {
    $.x = $x;
    $.y = $y;
  }
};

my $a = Point.new(x => 23, y => 42);

say $a.x; # 23

$a.set_coordinates(10, 20);

say $a.x; # 10

```

## 14.6. Inheritance of classes

\* Classes can inherit using the "is" keyword.

### Example 14-6. examples/classes/point\_06.p6

```
use v6;

class Point {
  has $.x is rw;
  has $.y is rw;

  method reset {
    $.x = 0;
    $.y = 0;
  }
};

class Point3D is Point {
  has $.z is rw;

  method reset {
    $.z = 0;
    Point.HOW.dispatch(self, 'reset');
    #self.WALK(:parent);
  }
}

my $a = Point3D.new(x => 23, y => 42, z => 12);

say $a.WHAT;
say $a.x;    # 23
say $a.z;    # 12

$a.reset;

say $a.x;    # 0
say $a.z;    # 0
```

## 14.7. Classes in Perl 6

### Example 14-7. examples/oop/class.p6

```
#!/usr/bin/perl6
use v6;

class Point {
  has $.x is rw;
  has $.y is rw;
```

```
method origo {
    $.x = 0;
    $.y = 0;
}
}

my $a = Point.new(x => 23, y => 42);
say $a.WHAT;    # Point

say $a.x;
say $a.y;

$a.origo;

say $a.x;
say $a.y;

$a.x = 19; # TODO why can I not ue $a.x(19) here ?

say $a.x;
```

# Chapter 15. Old slides

## 15.1. Why old slides?

These following slides were removed from the main body of the course but left here as they might be of some help. They might be out of date and incorrect.

## 15.2. Installing Pugs on Linux

Download Pugs from <http://www.pugscod.org/>

Ubuntu: **sudo aptitude install pugs**

Gentoo: **emerge pugs**

## 15.3. Building Pugs on Ubuntu

On Ubuntu it was quite straight forward, I think this is everything I needed:

```
sudo apt-get install subversion

# Installing GHC 6.4.2
wget http://haskell.org/ghc/dist/6.4.2/ghc-6.6-src.tar.bz2
wget
tar xjf ghc-6.4.2-src.tar.bz2
tar xjf
cd ghc-6.4.2
./configure --prefix=/opt/ghc66
make
sudo make install

mkdir ~/src
cd ~/src

# To compile Parrot
svn co https://svn.perl.org/parrot/trunk parrot
cd parrot
perl Configure.pl --prefix=$HOME/parrot --cc=cc --cxx=CC --link=cc --ld=cc
make
```

```
make test
make install

# added the following to ~/.bashrc and ran source ~/.bashrc
export PATH=$HOME/parrot/bin:$PATH
export LD_LIBRARY_PATH=$HOME/parrot/lib/
export PARROT_PATH=$HOME/work/parrot
export GHC=/opt/ghc642/bin/ghc

# To compile Pugs I use the following:
cd ~/src
svn co http://svn.pugscode.org/pugs
perl Makefile.PL
make

I have not "installed" it anywhere

# Now I can use it
./pugs -e 'say "Hello world"'
```

## 15.4. Running Pugs

Version information:

```
/usr/bin/perl6 -v
```

Oneliners:

```
/usr/bin/perl6 -e "say 42"
```

Help:

```
/usr/bin/perl6 -h
```

## 15.5. Division

There are other numerical operators as well, such as /, \*, -  
Especially / is interesting as 3 / 0 returns Inf.  
That is, in Perl6 the division by 0 problem is solved.

Comment: this is not true any more, it throws an exceptin now.

**Example 15-1. examples/scalars/divide.p6**

```
#!/usr/bin/perl6
use v6;

print "First number:";
my $a = =$*IN;
print "Second number:";
my $b = =$*IN;

my $c = $a / $b;

say "\nResult: $c";
```

# Table of Contents

#, 7  
\$, 9  
~, 11  
~=: 11  
~~, 39  
and, 15  
comments, 7  
else, 12  
eq, 14  
ge, 14  
given, 17  
gt, 14  
if, 12  
index, 18  
le, 14  
lt, 14  
ne, 14  
not, 15  
or, 15  
orelse, 15  
POD, 8  
print, 10  
prompt, 10  
regex, 39  
say, 9  
STDIN, 10  
substr, 18  
x, 12  
xor, 15