Perl 6 Cheat Sheet

SIGILS

```
$scalar
@array
%hash
&code
```

MAJOR/MINOR CONTEXTS

```
item   list       sink
 Str    flat/slice
 Num    lazy/eager/
 Bool   hyper/race
```

ACCESS ARRAYS HASHES

```
 whole:  @array[]      %hashe{}
element: @array[0]     %hash{'a'} or %hash<a>
  slice: @array[0,2]   %hash{'a','b'} or %hash<a b>
```

TWIGILS

```
$normal-lexical
$?compiler-constant
$*dynamic-or-global
$.public-accesor
$!private-accesor
$^positional-parameter
$:named-parameter
$=pod-info
$<named-match-capture>
$~slang-variable
```

COMPOSERS

```
[ ] array
{ } block/hash
< > quotewords
(,) list
:() signature
\() capture
```

AUTOMATIC DEREFERENCE

```
&($foo)(1,2)       == $foo(1,2)
@($foo)[1]         == $foo[1]
%($foo){'bar'}     == $foo<bar>
@(@($foo)[1])[2]   == $foo[1][2]
```

SPECIAL VARIABLES

```
$_       current topic
$/       regex result
$!       error object
@*ARGS   command line
@*INC    include path
%*ENV    environment
$*PID    process id
```

METAOPERATORS

```
[op]  reduce listop to A op B op C...
op=   A = A op B
!op   !(A op B)
»op«  hyper/vectorize
Zop   zip with operator op
Xop   cross with operator op
Rop   reverse args
Sop   sequentialize
```

OPERATOR DOMAINS

```
Numeric:  Stringy:  Value:
==        eq        eqv
!==(!=)   !eq(ne)   !eqv
+         ~
<         lt        before
>         gt        after
<=>       leg       cmp
<=        le        !after
>=        ge        !before

ObjectID: ===  !===
```

OPERATOR PRECEDENCE

tightest

```
.method .[] i
++ --
**
unary + - ~ ! ? ^
* / % %% div
+ -
x xx
~
&
| ^
sleep abs sin temp let
<=> leg cmp .. but
~~ > == gt eq === eqv !op
&&
|| ^^ // min max
??!! ff
= := op= =>
so not
, :
X Xop Z Zop ...
say die map etc.
and
or xor
<== ==>
```

loosest

SCOPE DECLARATORS

```
      my   lexical scope
     our   package scope
     has   instance scope
    anon   no scope at all
   state   persistent lexical
 augment   benign parasitic
supersede  deadly parasitic
```

CONTROL SYNTAX

```
for LIST { }                                    # explict $_ arg
for LIST -> $a, $b { }                          # explicit args
while/until EXPR { }                            # while TRUE/until True
repeat while/until EXPR { }                     # do at least once
loop { }   loop (a;b;c) { }                     # parens required!
if EXPR { } elsif EXPR { } else { }             # truthiness
with EXPR { } orwith EXPR { } else { }          # definedness
given EXPR { when EXPR { } default { } }        # switch statement
EXPR if EXPR for LIST;                          # list comprehension
next, last, redo                                # loop controls
proceed, suceed                                 # switch controls
```

REGEX METACHARACTERS

```
^ $      string begin/end
^^ $$    line begin/end
+        one or more
*        zero or more
?        zero or one
**1..3   repeat in range
()       capture
[]       no capture
<[]>     character class
<foo>    subrule
|        parallel or
||       serial or
<< >>    word boundary
```

REGEX MODIFIERS

```
:i      ignore case
:m      ignore marks
:g      global
:r      not backtracking
:s      significant whitespace
:4th    nth occurrence
:4x     n times
```

REGEX CHARACTER CLASSES

```
.  == any character, \N non \n
\s == <space>, \S non
\d == <digit>, \D non
\w == <+alpha+digit+[_]>
```

BASIC TYPES

```
AST Any Mu Label Block Code Callable CallFrame Int atomicint Rat
FatRat Num Rational Real Complex Numeric Str Cool IntStr RatStr NumStr
ComplexStr Stringy Date DateTime Duration Instant Dateish Sub Routine
Macro Method Submethod Variable Signaturae Parameter Whatever
HyperWhatever WhateverCode Proxy ObjAt Version Nil Bool Junction Scalar
```

QUOTING CONSTRUCTS

Data

```
my ($a, $o, $h) = (3, 4, 5);
my @let = 'a', 'b';
my %bin = o => True, t => False;
sub greetings { say "Hello" }
```

Literal Strings

Strings are usually represented using some form of quoting construct. The most minimalistic is **Q**, which can be used with the shortcut ⌈...⌋ or via **Q** inmediately followed by almost any of pair of delimiters surrounding the text.

```
  Q[A literal string]
  ⌈Another literal string⌋
  Q{Almost any delimiter!}
✘ Q(Interpreted as a function call)
✘ Q'This will not work'
✔ Q (This works though!)
✔ Q 'This one too!'
```

Neither () nor '' can be used as regular delimiters with **Q**. To use them, a space must separate them from **Q**, **q**, and **qq**.

Q with any pair of delimiters can be used to quote strings verbatim without interpolation or escaping. Nonetheless, its behavior can be changed through the used of adverbs.

Basic Features

ADVERBS

short	long	meaning	example
:q	:single	Interpolate \\, \qq and delimiter escaping with \	Q:q[\s] == Q:q[\\s]
:qq	:double	Interpolate with :s, :a, :h, :f, :c, :b	Q:qq!$a!
:s	:scalar	Interpolate $ vars	Q:s^$c^
:a	:array	Interpolate @ vars	Q:a (@let => @let[])
:h	:hash	Interpolate % vars	Q:h!%bin => bin{}!
:f	:function	Interpolate & calls	Q:f!&greet()!
:c	:closure	Interpolate {...} expressions	Q:c/3{$a**2 + $o**2}/
:b	:backslash	Enable backslash escapes (\n, \qq, \$foo, etc)	Q:b[\nHi]

Advanced Features

ADVERBS

short	long	meaning	example
:x	:exec	Execute as command and return results	Q:x{date}
:w	:words	Split result on words (no quote protection)	.say for Q:w{'a b' c}
:ww	:quotewords	Split result on words (with quote protection)	.say for Q:ww{'a b' c}
:v	:val	Convert to allomorph if possible	Q:v{42}.^name
:to	:heredoc	Parse result as heredoc terminator	Q:to{END}; Content starts in next line END

Few Shortcuts

adverb form	shortcut	usual form	meaning
Q:q[...]	q[...]	'...'	same as before
Q:qq[...]	qq[...]	"..."	same as before
Q:q:w[...]	qw[...]	<...>	word quoting
Q:q:ww[...]	qww[...]		word quoting with quote protection
Q:qq:w[...]	qqw[...]		word quoting with interpolation
Q:qq:ww[...]	qqww[...]	<<...>> or « »	word quoting with interpolation and quote protection
Q:q:x[...]	qx[...]		shell quoting
Q:qq:x[...]	qqx[...]		shell quoting with interpolation

RESOURCES

LINKS

perl.org
rakudo.org

IRC -- irc.freenode.net

#perl6
#perl6-dev
#moarvm

SOCIAL

/r/perl6
@perl6org