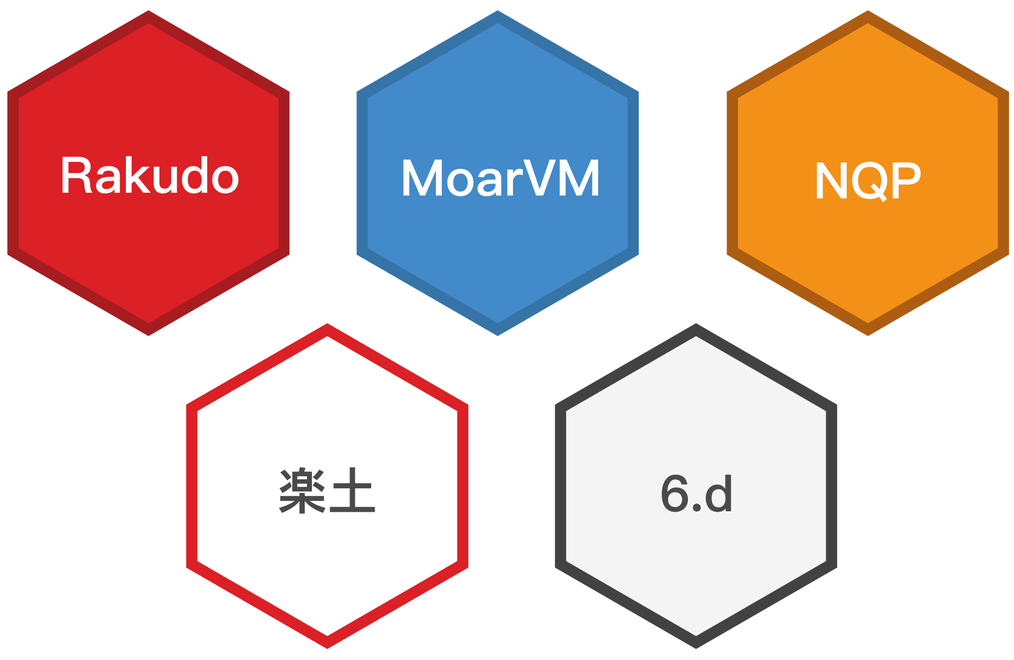
Perl 6 ESSENTIALS CHEET SHEET

Perl6.org

INSTALL A MODULE

Using Zef package tool:

```
$ zef search CSV
$ zef info CSV::Parser
$ zef install CSV::Parser      Installs module
$ zef upgrade CSV::Parser      Upgrade module
```



Hyper Operator

```
my @prefixes  = 'A'..'E';
my @roots     = 1, 2;
my @postfixes = 'a'..'e';

say @roots.map: ~(@prefixes »~» * «~« @postfixes);

# Output: (A1a B1b C1c D1d E1e A2a B2b C2c D2d E2e)
```

Subset

```
subset Length8      of Str where *.chars < 8;
subset UpCase       of Str where none('A'..'Z') ∈ *.comb.Set;
subset LowerCase    of Str where none('a'..'z') ∈ *.comb.Set;
subset IntNumber    of Str where none('0'..'9') ∈ *.comb.Set;

my $guess = prompt('Enter your password:');

given $guess {
    when Length8     { say '密码长度必须为 8 位 以上'; proceed }
    when  UpCase     { say '密码必须包括大写字母';     proceed }
    when LowerCase { say '密码必须包含小写字母';     proceed }
    when IntNumber  { say '密码必须包含数字';                 }
}
```

Feed

```
# Feed (left-to-right) with parentheses, read top-to-bottom
my @result = (
    <people of earth>     # (1) Start with the input
    ==> map({ .tc })      # (2) Capitalize the words
    ==> grep /<[PE]>/     # (3) Look for P or E
    ==> sort              # (4) Sort, result is <Earth People>
);

say @result;


# To assign without parentheses, use another feed operator
my @result
    <== sort()                  # (4) Sort, result is <Earth People>
    <== grep({ /<[PE]>/ })      # (3) Look for P or E
    <== map({ .tc })            # (2) Capitalize the words
    <== <people of earth>;  # (1) Start with the input
```

Grammars

```
grammar Parser {
    rule  TOP  { I <love> <lang> }
    token love { '♥' | love }
    token lang { < Perl Rust Go Python Ruby > }
}

say Parser.parse: 'I ♥ Perl';
# OUTPUT: 「I ♥ Perl」 love => 「♥」 lang => 「Perl」

say Parser.parse: 'I love Rust';
# OUTPUT: 「I love Rust」 love => 「love」 lang => 「Rust」
```

Junctions

```
my @valid = <foo bar baz>;
my $what = 'ber';
say "$what is not valid" if $what eq none @valid;
say "A ber or a bar"     if $what eq 'ber' | 'bar';
```

Pattern Matching

```
# 使用签名进行模式匹配

class Body   { has ( $.head, @.arms, @.legs ) }
class Person { has ( $.mom,  $.body, $.age  ) }

my $age = 42;
multi person's-age-and-legs (
    Person ( :$age where * > 40,
             :$body ( :@legs, *% ),
             *%
           )
    ) { say "$age {+@legs}" }

person's-age-and-legs Person.new(
    :$age,
    body => Body.new(
        :head,
        :2arms,
        legs => <left middle right>
    )
);

# 42 3
```

Chained Operators

```
for (-25..25) X (-25..25) -> ($y, $x) {
    print (16² < $x² + $y² < 20²) ?? 'o' !! '.';
    $x==25 && say '';
}
```

Muti Dispatch

```
multi sub greet($name) {
    say "Ahoj, $name!";
}
multi sub greet($name, $greeting) {
    say "$greeting, $name!";
}
greet('Anna'); # Ahoj Anna
greet('Лена', 'Привет '); # Привет, Лена"
```