

# Macros

masak

GPW 2012

2012-03-06

**What are macros?**

# Dive-through example

```
constant LOGGING = False;
```

```
sub LOG($message) {  
  if LOGGING {  
    $*ERR.say: $message;  
  }  
}
```

```
LOG "The value is: {hard-to-compute()}";
```

```
constant LOGGING = False;
```

```
macro LOG($message) {
```

```
  if LOGGING {
```

```
    quasi {
```

```
      $*ERR.say: {{{$message}}};
```

```
    }
```

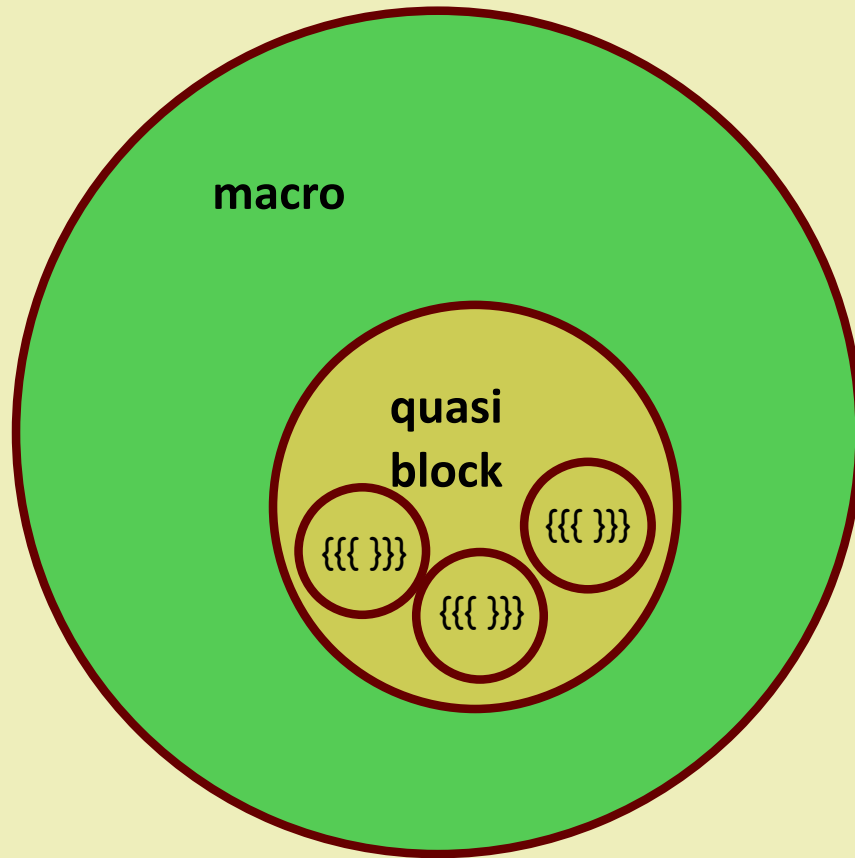
```
  }
```

```
}
```

```
LOG "The value is: {hard-to-compute()}";
```

**hi, I'm masak**

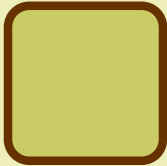
**What are macros?**



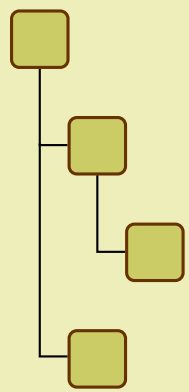


**Subroutines map values**

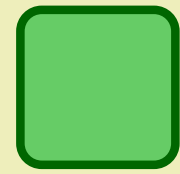
**Macros map code**



Source code



Abstract  
Syntax  
Tree



Target code

"Meta" code

*to talk about **code***

Normal code

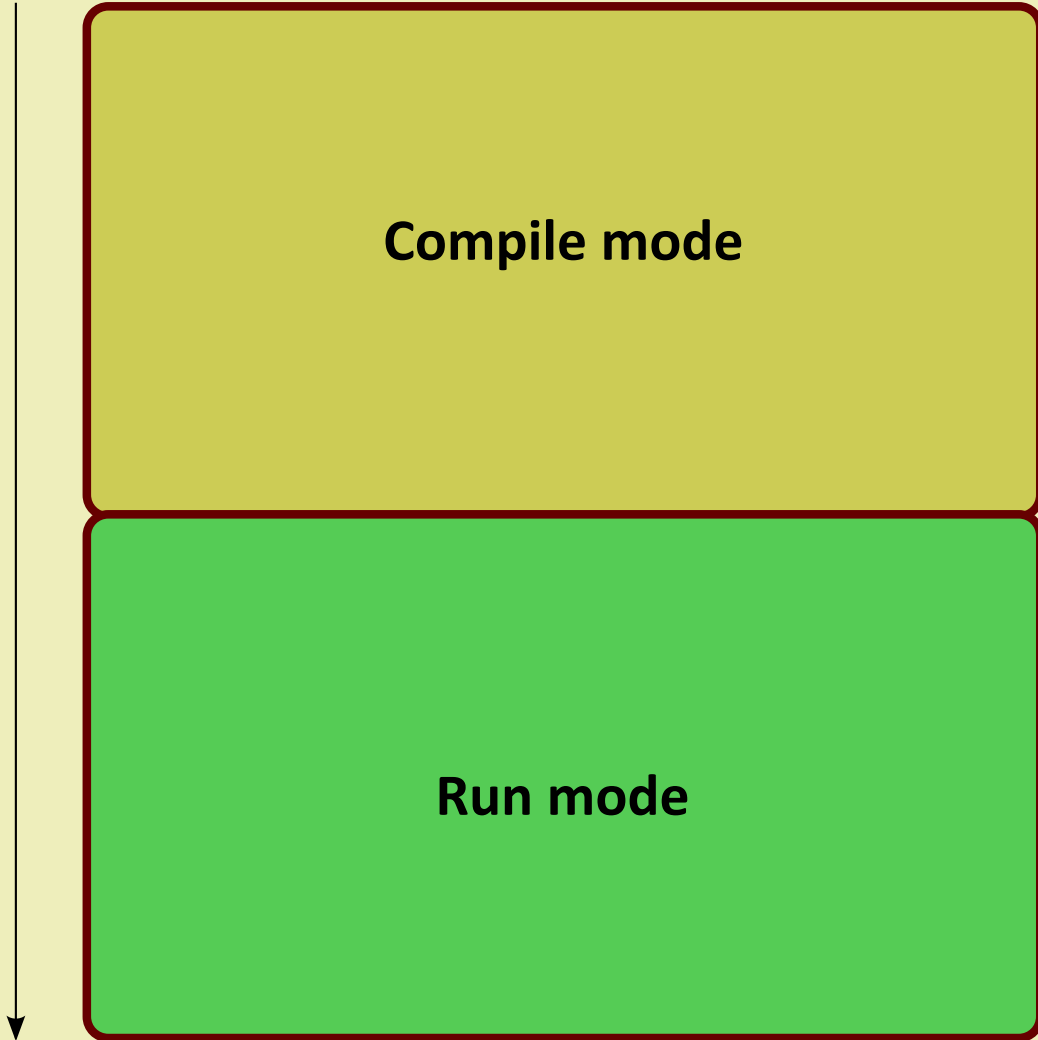
*to talk about **values***

# Perl 6: modable

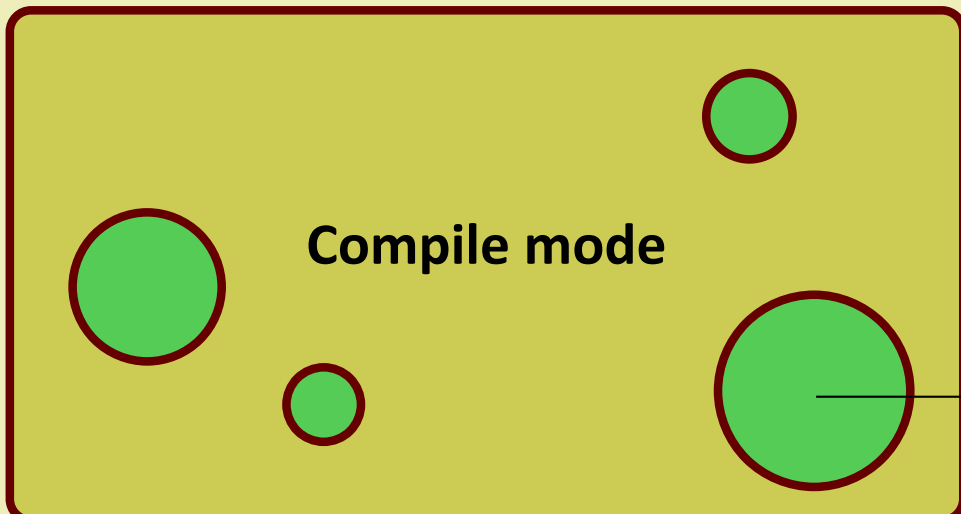
time

**Compile mode**

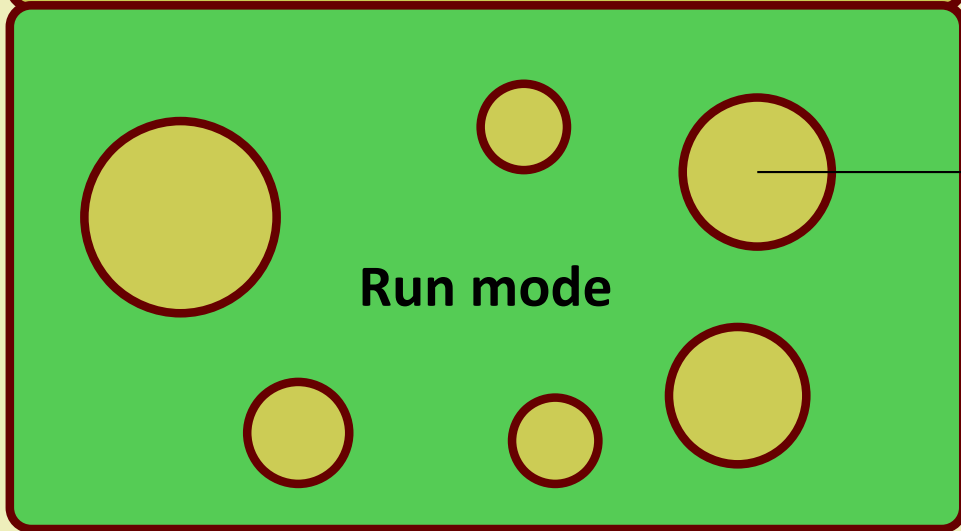
**Run mode**



time



BEGIN semantics



eval semantics

```
BEGIN { say "OH HAI" }  
my $value = eval $code;
```

```
constant ANSWER = 42;  
class Dugong {  
    has $.nose = calculate-nose();  
}
```

```
calculation1() && calculation2();
```

**Compile time:**

**BEGIN -- CHECK**

**Run time:**

**INIT -- START -- END**

**Blocks:**

**ENTER -- LEAVE**

**PRE -- POST**

**Loops:**

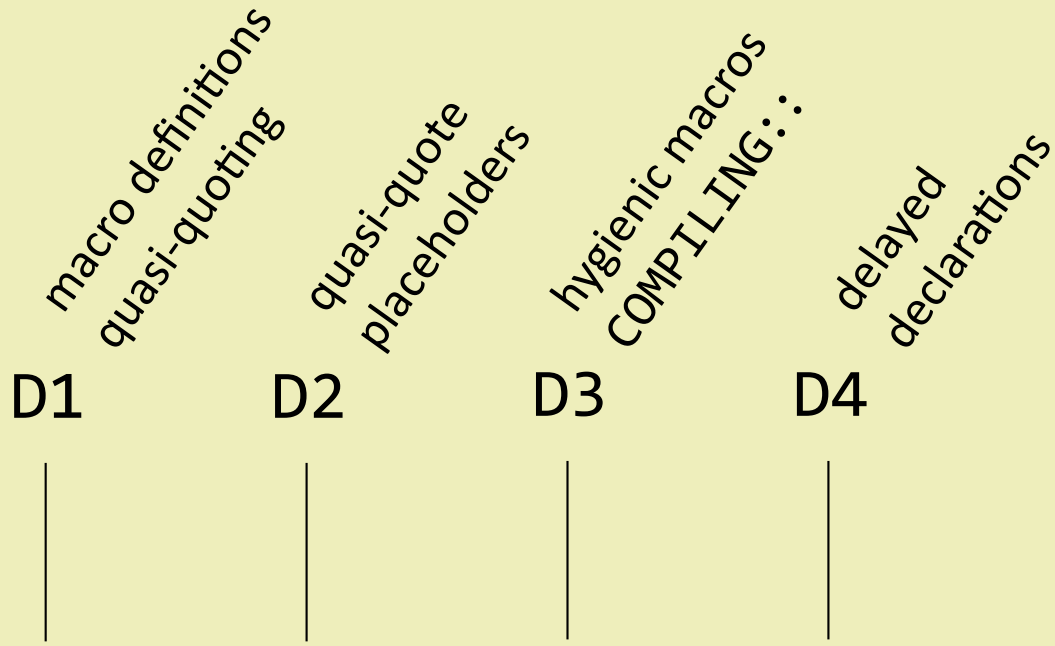
**FIRST -- NEXT -- LAST**



**Macros grant**

**I remember there**

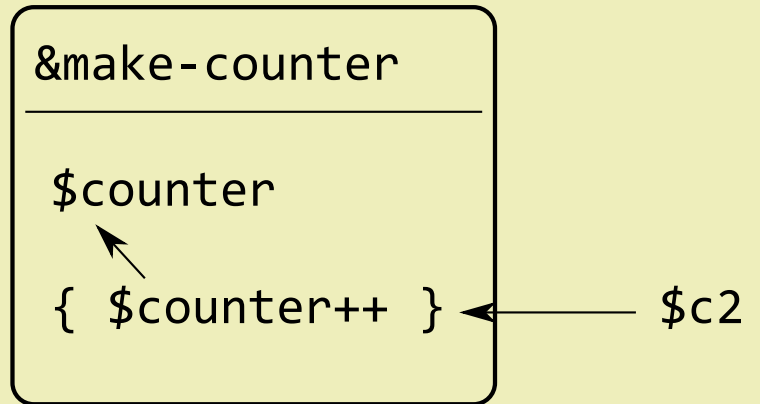
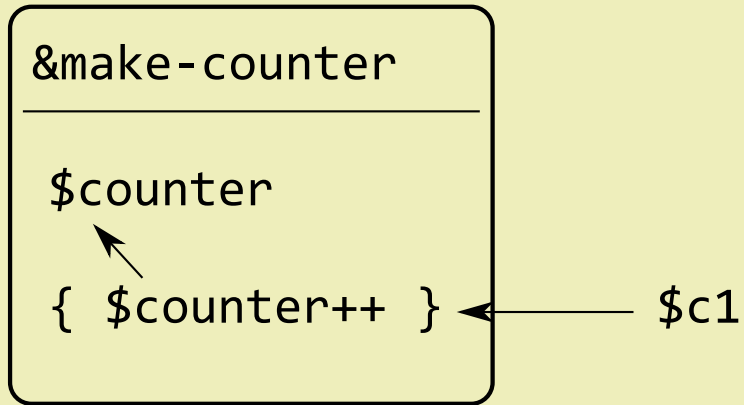
**being lots of beer**



**Closures are wonderful**

```
sub make-counter($init) {  
  my $count = $init;  
  return { $count++ };  
}
```

```
my $c1 = make-counter(5);  
my $c2 = make-counter(42);
```



```
class Rectangle {  
  has $.width;  
  has $.height;  
  
  method area { $.width * $.height }  
  
  method circumference {  
    $.width + $.height  
    + $.width + $.height  
  }  
}
```

```
sub make-rectangle($width, $height) {  
  return sub($_) {  
    when 'area' { $width * $height }  
    when 'circumference' {  
      $width + $height  
      + $width + $height  
    }  
  }  
}
```



**Macros, bullet time**

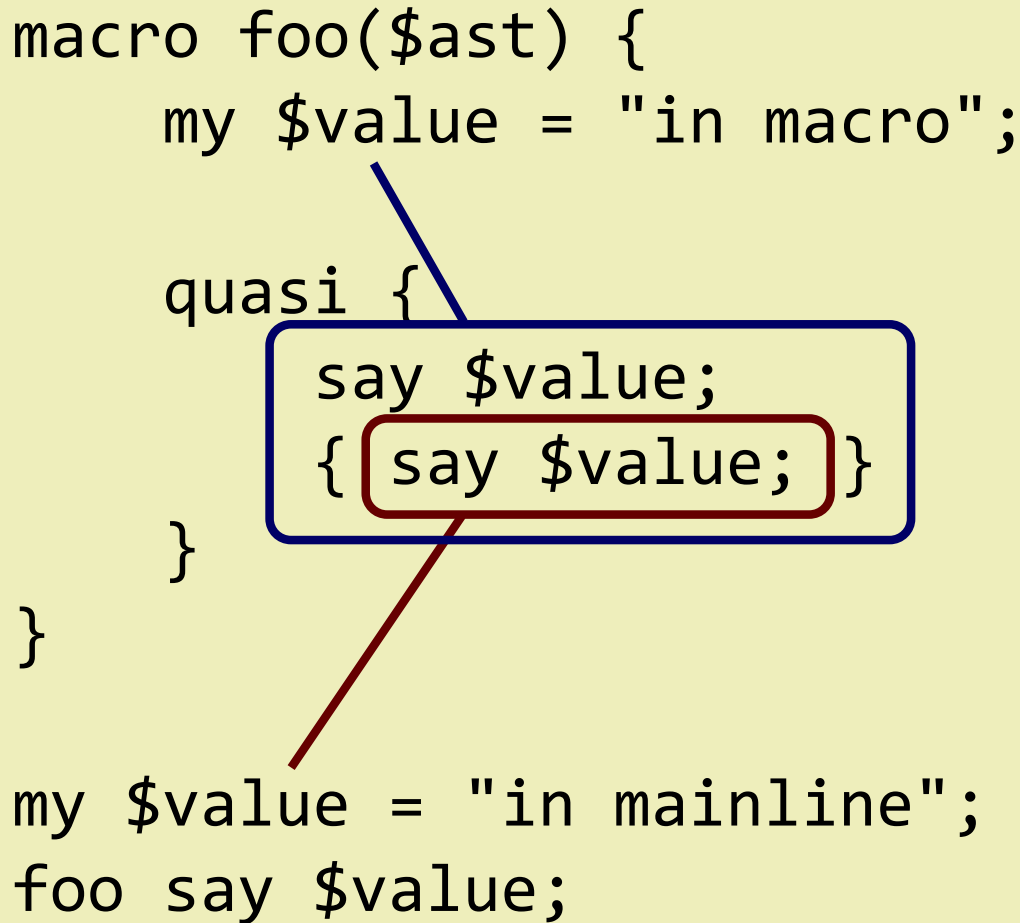
```
macro foo($ast) {  
    my $value = "in macro";  
  
    quasi {  
        say $value;  
        {{{$ast}}};  
    }  
}
```

```
my $value = "in mainline";  
foo say $value;
```

```
macro foo($ast) {  
    my $value = "in macro";  
  
    quasi {  
        say $value;  
        {{{$ast}}};  
    }  
}
```

```
my $value = "in mainline";  
foo say $value;
```

```
macro foo($ast) {  
    my $value = "in macro";  
  
    quasi {  
        say $value;  
        { say $value; }  
    }  
}  
  
my $value = "in mainline";  
foo say $value;
```



```
macro foo($ast) {  
    my $value = "in macro";  
  
    quasi {  
        say $value;  
        { say $value; }  
    }  
}  
  
my $value = "in mainline";  
{  
    say $value;  
    { say $value; }  
}
```

The diagram illustrates the scope of variables in a Perl macro. A blue line connects the `my $value` declaration in the macro to the `say $value` statement within the macro's `quasi` block, indicating that the macro's local variable is used. A red line connects the `my $value` declaration in the mainline to the `say $value` statement within the mainline's block, indicating that the mainline's variable is used. Red boxes highlight the `say $value;` lines in both the macro and mainline, while blue boxes highlight the entire `quasi` block and the mainline block.

**There is no spoon**

```
my $x;
```

```
role R {  
  method foo { $x }  
}
```

```
class C does R {  
}
```

*“ We need a solution  
that makes us  
need less vodka.”*

-- jnthn



# Conclusion

**Macros rock**

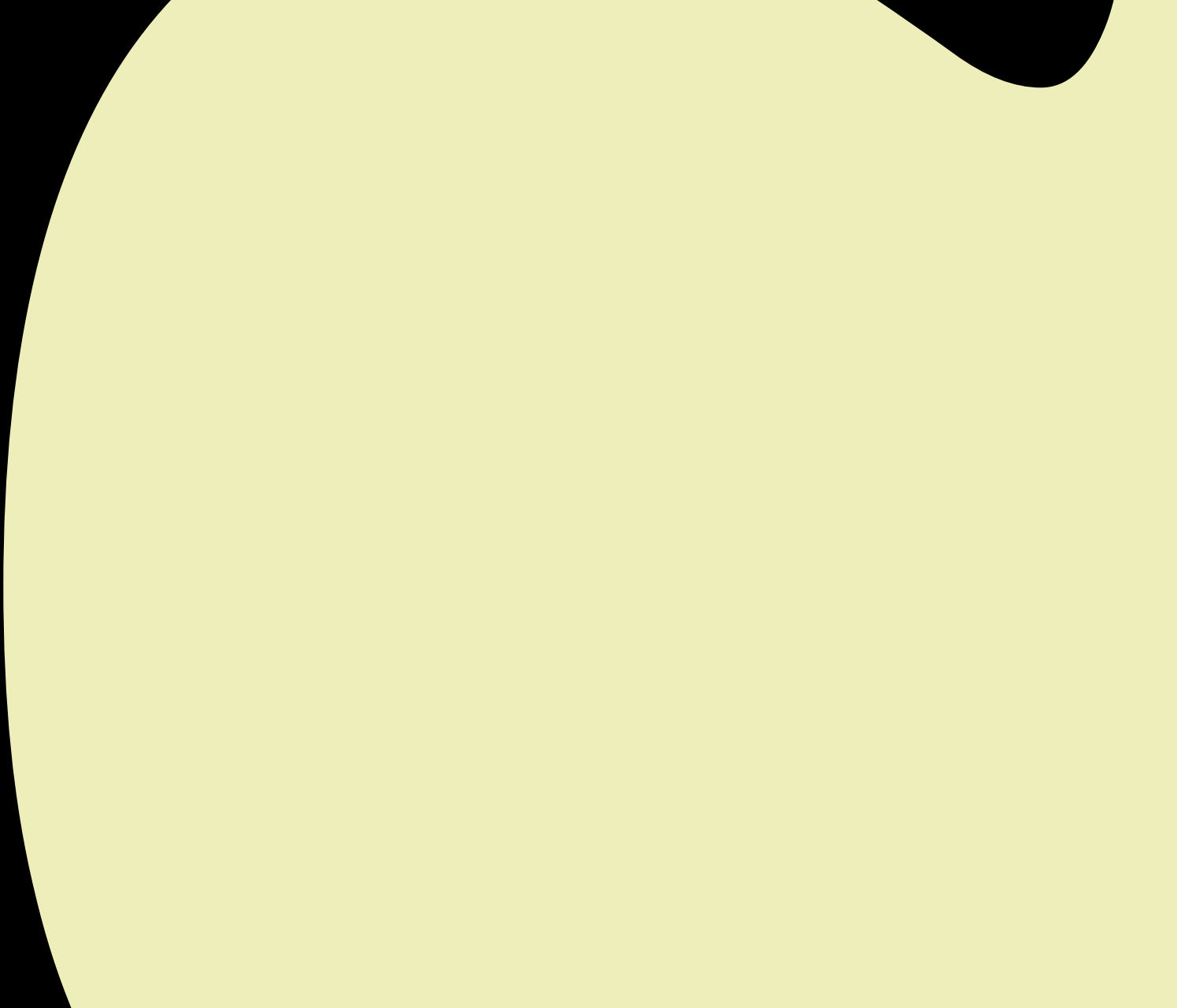
**Implementing macros**

**is interesting**

**Stand by  
for macros  
in Rakudo**











**acrr**

**Macros**

# Macros

masa

GPW 201

2012-03-0

Conclusion

There is no spoon

What are macros?

hi, I'm

macros, bullet time

# Macros

What are macros?

masak

GPW 2012

2012-03-06

Closures are wonderful

Perl 6: modable

Macros grant

```
my $x;
rule R {
  method foo { $x }
}
class C does R {
}
```

Macros rock

is interesting

```
constant LOGGING = False;
sub LOG($message) {
  if LOGGING {
    $*ERR.say: $message;
  }
}
LOG "The value is: {hard-to-compute()}";
```

Dive-through example

## Conclusion

There is no spoon

What are macros?

hi, I'm masak

```
macro foo($ast) {
  my $value = "in macro";
  quasi {
    say $value;
    {{{$ast}}};
  }
}
my $value = "in mainline";
foo say $value;
```

```
in macro";
value;
in_mainline";
```

Macros, bullet time

# Macros

What are macros?



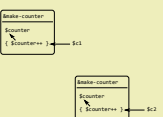
masak  
GPW 2012  
2012-03-06

Closures are wonderful

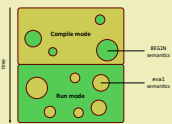
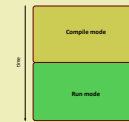
Macros grant

Perl 6: modable

```
sub make-counter($init) {
  my $count = $init;
  return { $count++ };
}
my $c1 = make-counter(5);
my $c2 = make-counter(42);
```



I remember there  
being lots of beer



"We need a solution  
that makes us  
need less vodka."  
- path

Stand by

for macros

Implementing macros  
in Rakudo

is interesting

Macros rock

Dive-through example

```
my $x;

sub foo {
  method foo { $x }
}

class C {
  sub $x {
  }
}
```

```
constant LOGGING = False;

sub LOG($message) {
  if LOGGING {
    say $message;
  }
}

LOG "The value is: {hard-to-compute[1]}";
```

```
constant LOGGING = False;

macro LOG($message) {
  if LOGGING {
    say $message;
  }
}

LOG "The value is: {hard-to-compute[1]}";
```

### Conclusion

There is no spoon

What are macros?

hi, I'm masak

Subroutines map values

Macros map code



"Masak" code to talk about code

Normal code to talk about values

Macros, bullet time

# Macros

What are macros?

masak  
GPW 2012  
2012-03-06

```
macro foo($x) {
  my $code = "in macro";
  quote {
    say $code;
    {{ $x }};
  }
  my $code = "in machine";
  Foo say $code;
}
```

```
macro foo($x) {
  my $code = "in macro";
  quote {
    say $code;
    {{ $x }};
  }
  my $code = "in machine";
  Foo say $code;
}
```

```
macro foo($x) {
  my $code = "in macro";
  quote {
    say $code;
    {{ $x }};
  }
  my $code = "in machine";
  Foo say $code;
}
```

```
macro foo($x) {
  my $code = "in macro";
  quote {
    say $code;
    {{ $x }};
  }
  my $code = "in machine";
  Foo say $code;
}
```

Closures are wonderful

Macro grant

Perl 6: modable

I remember there  
being lots of beer



```
MASAK { say "in MASK" };
my $code = eval $code;
constant ANOTHER = 42;
class Number {
  has $-base = calculate-base();
  calculate() { 88 calculate-base();
}
```

```
Compile time:
MASAK -- CODE

Run time:
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
MASK -- STATE -- END
```

```
class Rectangle {
  has $width;
  has $height;
  method area { $width * $height };
  method circumference {
    $width * 2 + $height * 2
  };
}
```

```
class Rectangle {
  has $width;
  has $height;
  method area { $width * $height };
  method circumference {
    $width * 2 + $height * 2
  };
}
```

```
sub make-counter($init) {
  my $count = $init;
  return { $count++ };
}

my $c1 = make-counter(5);
my $c2 = make-counter(42);
```

"We need a solution  
that makes us  
need less vodka."  
- path

Stand by

for macros

Implementing macros  
in Rakudo

is interesting

Macros rock

Dive-through example

```
my $x;

sub foo {
    method foo { $x }
}

class C {
    sub foo {
        my $x = 1;
    }
}

my $obj = C.new;
$obj.foo;
```

```
constant LOGGING = False;

sub LOG($message) {
    if LOGGING {
        say "LOG: $message";
    }
}

LOG "The value is: {hard-to-compute[1]}";
```

```
constant LOGGING = False;

macro LOG($message) {
    if LOGGING {
        say "LOG: $message";
    }
}

LOG "The value is: {hard-to-compute[1]}";
```

## Conclusion

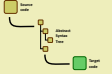
There is no spoon

What are macros?

hi, I'm masak

Subroutines map values

Macros map code



"Meta" code to talk about code  
Normal code to talk about values

Macros, bullet time

# Macros

What are macros?

masak  
GPW 2012  
2012-03-06

```
macro foo($x) {
    my $code = "in macro";
    quote {
        say $code;
        ((($x)));
    }
}

my $code = "in mainline";
foo say $code;
```

```
macro foo($x) {
    my $code = "in macro";
    quote {
        say $code;
        ((($x)));
    }
}

my $code = "in mainline";
foo say $code;
```

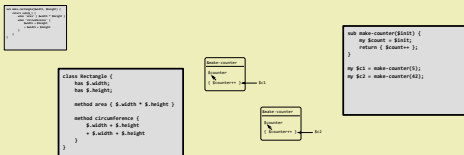
```
macro foo($x) {
    my $code = "in macro";
    quote {
        say $code;
        ((($x)));
    }
}

my $code = "in mainline";
foo say $code;
```

Closures are wonderful

Macro grant

Perl 6: modable



I remember there  
being lots of beer



```
my $obj = Rectangle.new(10, 5);
$obj.width;
```

```
my $obj = Rectangle.new(10, 5);
$obj.height;
```

```
my $obj = Rectangle.new(10, 5);
$obj.circumference;
```

```
Compile-time:
my $obj = Rectangle.new(10, 5);
my $obj.width;
```

```
Run-time:
my $obj = Rectangle.new(10, 5);
my $obj.height;
```

```
Loop-time:
my $obj = Rectangle.new(10, 5);
my $obj.circumference;
```