Perl 6.d (Diwali)

await no longer blocks a thread while waiting

```
await ^10 .map: { start await ^25 .map: { start await Promise.in: 1 } }
say "It took $(now - ENTER now) seconds to run";
```

easier subclassing of IO::Handle type

```
class IO::URL is IO::Handle {
    has $.URL is required;
    has Buf $!content;

    submethod TWEAK {
        use WWW;       # ecosystem module for web page fetching
        use DOM::Tiny; # ecosystem module for HTML parsing
        $!content := Buf.new:
            DOM::Tiny.parse(get $!URL).all-text(:trim).encode;
        self.encoding: 'utf8';
    }
    method READ(\bytes) { $!content.splice: 0, bytes }
    method EOF { not $!content }
}

my $fh := IO::URL.new: :URL<www.perl6.org>;

# .slurp and print all the content from the website. We can use all other
# read methods, such as .lines, or .get, or .readchars. All of them work
# correctly, even though we only defined .READ and .EOF
$fh.slurp.say;
```

smarter $*ARGFILES in sub MAIN

```
sub MAIN($separator) {
    say "Enter words to separate with `$separator`:";
    say "Result is: ", words.join: " $separator "
}
```

```
$ perl6 script.p6 et
Enter words to separate with `et`:
un deux
trois
Result is: un et deux et trois
```

Thread-Safe, Atomic Operations

```
# Regular ops: not thread safe! Wrong result!
my int $total = 0;
await start { for ^20000 { $total++ } } xx 10;
say $total; # OUTPUT: «188758␤»

# Using atomic operations: thread-safe and correct
my atomicint $atotal = 0;
await start { for ^20000 { $atotal⚛++ } } xx 10;
say $atotal; # OUTPUT: «200000␤»
```

Set Operator Improvements

```
say <a b c>.Set (-) <d e f b>.Bag
# OUTPUT: «Bag(a, c)»
```