functional programming you already know

@KevlinHenney

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

A Pattern Language for Distributed Computing

Volume 4

Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

WILEY SERIES IN
SOFTWARE DESIGN PATTERNS

PATTERN-ORIENTED SOFTWARE ARCHITECTURE

On Patterns and Pattern Languages

Volume 5

Frank Buschmann
Kevlin Henney
Douglas C. Schmidt

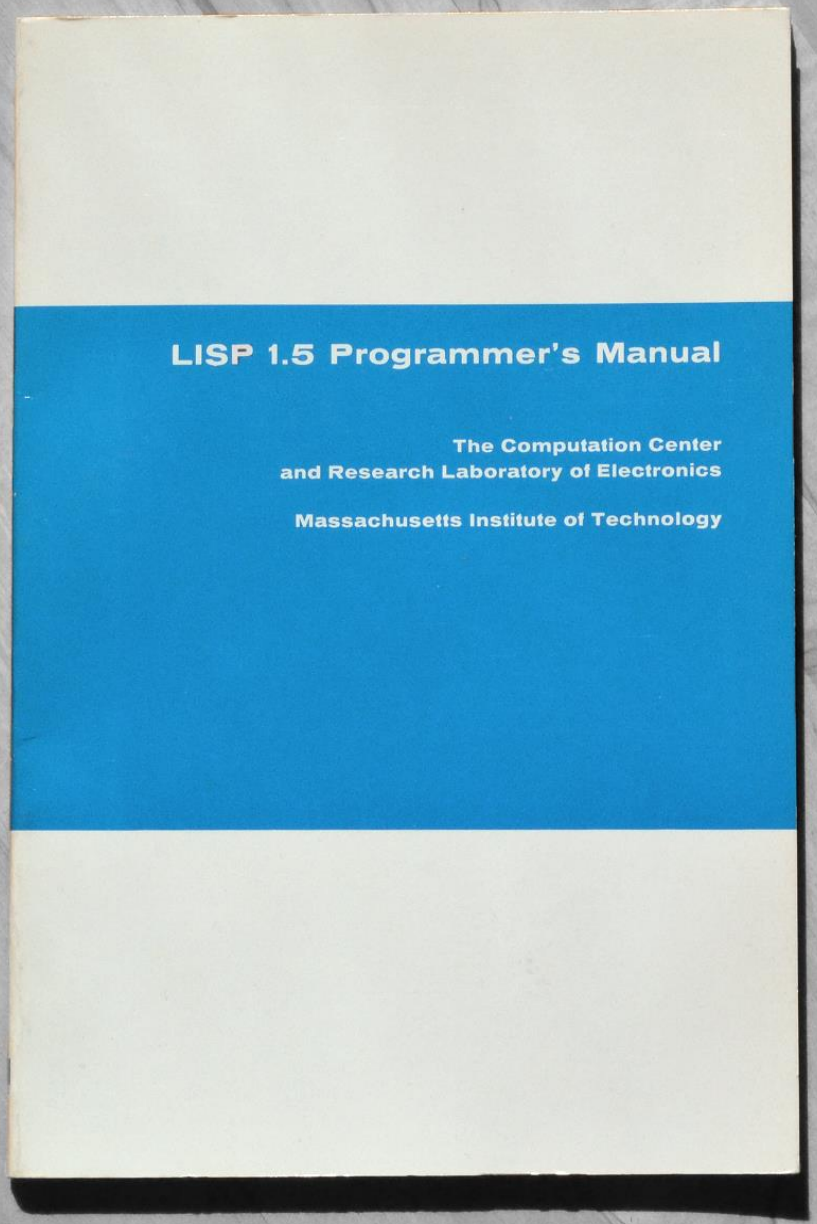

Kevlin Henney 编
李军 译 吕骏 审校
電子工業出版社
PUBLISHING HOUSE OF ELECTRONICS INDUSTRY
http://www.phei.com.cn
O'REILLY
オライリー・ジャパン
97
Collective Wisdom
from the Experts
97 Things Every
Programmer
Should Know
O'REILLY
Edited by Kevlin Henney

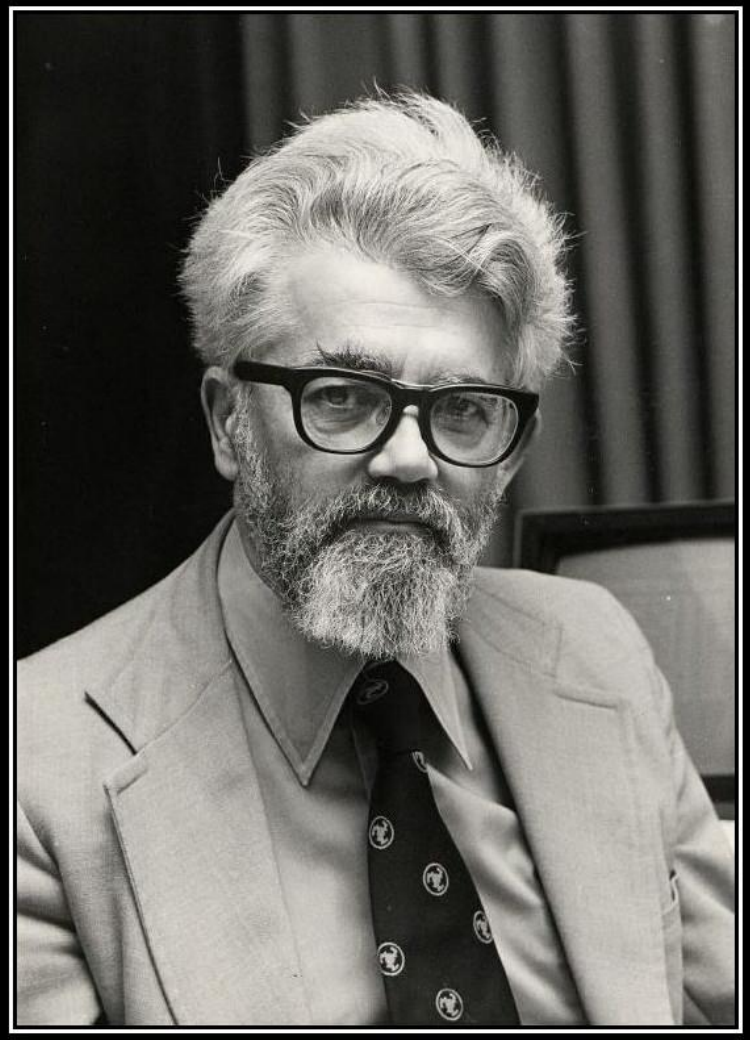

Richard A. Lanham
Adam Jacot de Boinod
I NEVER KNEW THERE WAS A WORD FOR IT
'Very funny'
Independent on Sunday
STEPHEN FRY
Michael Quinion
Joie de vivre n.
COMPILED BY ADRI
WORDSWORTH REFERENCE
The Wordsworth Book of Intriguing Words
REFERENCE
English mology
T. F. HOAD
word origins
the language report
A COMPREHENSIV WORD-FINDING DICTIONARY
Chambers
BILL BRYSON
TROUBLESOME WORDS
'Combines the virtues of a first-class work of reference with the pleasure of a good read' The Times
FULLY REVISED AND UPDATED
COLLINS REFERENCE DICTIONARY
MATHEMATICS
E. J. BOROWSKI AND J. M. BORWEIN
Oxford
Samuel Johnson's
LOCK
LONG WORDS
ER ME
REPENT! THE PREPOSITION IS NEAR!
JEFFREY KACIRK
AUTHOR OF Forgotten English
/ WordFriday

code-switching, *noun*

- Switching between two or more languages or varieties, such as dialects, in the course of a conversation.
- A code-switch can occur within a sentence, sometimes multiple times, or at a more natural break within a conversation.

https://www.facebook.com/WordFriday/posts/594158534005441


LISP 1.5 Programmer's Manual
The Computation Center
and Research Laboratory of Electronics
Massachusetts Institute of Technology



PROGRAMMING

You're Doing It Completely Wrong.

I believe that the current state of the art of computer programming reflects inadequacies in our stock of paradigms, in our knowledge of existing paradigms, in the way we teach programming paradigms, and in the way our programming languages support, or fail to support, the paradigms of their user communities.

The Paradigms of Programming

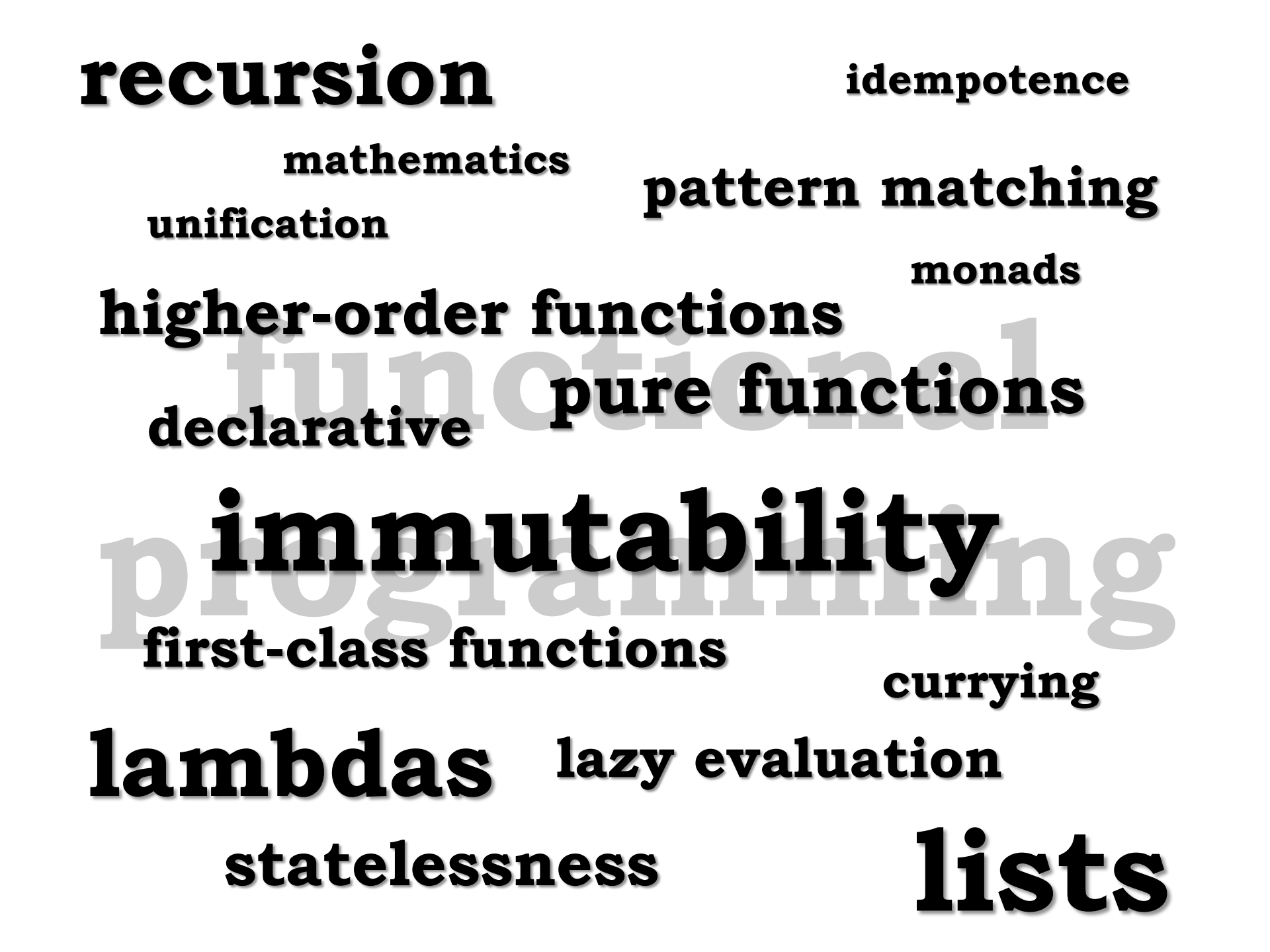
Robert W. Floyd
Stanford University

Paradigm(pæ·radim, −dəim) . . . [a. F. *paradigme*, ad. L. *paradigma*, a. Gr. παραδειγμα pattern, example, f. παραδεικνυ·ναι to exhibit beside, show side by side. . .]
1. A pattern, exemplar, example.
1752 J. Gill *Trinity* v. 91
The archetype, paradigm, exemplar, and idea, according to which all things were made.
From the Oxford English Dictionary.

Today I want to talk about the paradigms of programming, how they affect our success as designers of computer programs, how they should be taught, and how they should be embodied in our programming languages.

A familiar example of a paradigm of programming is the technique of *structured programming*, which appears to be the dominant paradigm in most current treatments of programming methodology. Structured programming, as formulated by Dijkstra [6], Wirth [27, 29], and Parnas [21], among others, consists of two phases.

In the first phase, that of top-down design, or stepwise refinement, the problem is decomposed into a very small number of simpler subproblems. In programming the solution of simultaneous linear equations, say, the first level of decomposition would be into a stage of triangularizing the equations and a following stage of back-substitution in the triangularized system. This gradual decomposition is continued until the subproblems that arise are simple enough to cope with directly. In the simultaneous equation example, the back substitution process would be further decomposed as a backwards iteration of a process which finds and stores the value of the *i*th variable from the *i*th equation. Yet further decomposition would yield a fully detailed algorithm.


recursion
idempotence
mathematics
pattern matching
unification
monads
higher-order functions
functional
pure functions
declarative
immutability
programming
first-class functions
currying
lambdas
lazy evaluation
statelessness
lists

Fizz buzz is a group word game for children to teach them about division.

http://en.wikipedia.org/wiki/Fizz_buzz

Adults may play Fizz buzz as a drinking game, where making a mistake leads to the player having to make a drinking-related forfeit. [*citation needed*]

http://en.wikipedia.org/wiki/Fizz_buzz

Fizz buzz has been used as an interview screening device for computer programmers.

http://en.wikipedia.org/wiki/Fizz_buzz

```
=IF(AND(MOD(ROW(),3)=
0,MOD(ROW(),5)=0),"Fizz
Buzz",IF(MOD(ROW(),3)=
0,"Fizz",IF(MOD(ROW(),5)
=0,"Buzz",ROW())))
```

Excel is the world's most popular functional language.

Simon Peyton-Jones

```
=IF(AND(MOD(ROW(), 3) = 0,
        MOD(ROW(), 5) = 0),
    "FizzBuzz",
    IF(MOD(ROW(), 3) = 0,
        "Fizz",
        IF(MOD(ROW(), 5) = 0,
            "Buzz",
            ROW())))
```

To iterate is human,
to recurse divine.

— L Peter Deutsch

```
int factorial(int n)
{
    int result = 1;
    while(n > 1)
        result *= n--;
    return result;
}
```

```
int factorial(int n)
{
    if(n > 1)
        return n * factorial(n - 1);
    else
        return 1;
}
```

```
int factorial(int n)
{
    return
        n > 1
        ? n * factorial(n - 1)
        : 1;
}
```

$$n! = \begin{cases} 1 & \text{if } n = 0, \\ (n-1)! \times n & \text{if } n > 0. \end{cases}$$

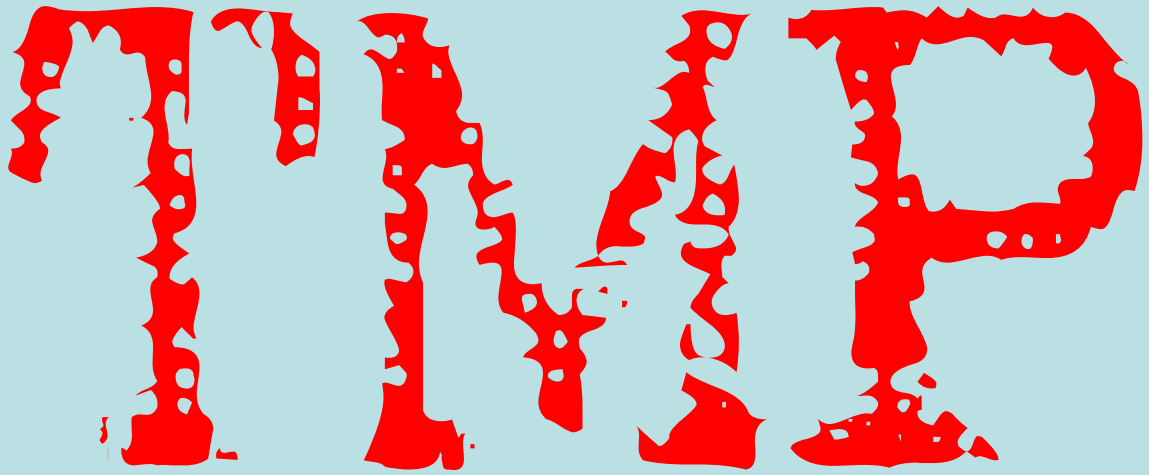
$$n! = \prod_{k=1}^{n} k$$

```
seriesProduct(k, k, 1, n)
```

```
reduce(
    lambda l, r: l*r,
    range(1, n+1), 1)
```

```
reduce(
    operator.mul,
    range(1, n+1), 1)
```

```
// Erwin Unruh, untitled program,
// ANSI X3J16-94-0075/ISO WG21-462, 1994.

template <int i>
struct D
{
    D(void *);
    operator int();
};

template <int p, int i>
struct is_prime
{
    enum { prim = (p%i) && is_prime<(i>2?p:0), i>::prim };
};

template <int i>
struct Prime_print
{
    Prime_print<i-1>    a;
    enum { prim = is_prime<i,i-1>::prim };
    void f() { D<i> d = prim; }
};

struct is_prime<0,0> { enum { prim = 1 }; };
struct is_prime<0,1> { enum { prim = 1 }; };
struct Prime_print<2>
{
    enum { prim = 1 };
    void f() { D<2> d = prim; }
};

void foo()
{
    Prime_print<10> a;
}

// output:
// unruh.cpp 30: conversion from enum to D<2> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<3> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<5> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<7> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<11> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<13> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<17> requested in Prime_print
// unruh.cpp 30: conversion from enum to D<19> requested in Prime_print
```


TMP

```
struct Fizz{};
struct Buzz{};
struct FizzBuzz{};

template<int i>
struct RunFizzBuzz
{
   typedef vector<int_<i> > Number;

   typedef typename if_c<(i % 3 == 0) && (i % 5 == 0), FizzBuzz,
                     typename if_c<i % 3 == 0, Fizz,
                              typename if_c<i % 5 == 0, Buzz, Number>::type>::type >::type t1;

   typedef typename push_back<typename RunFizzBuzz<i - 1>::ret, t1>::type ret;
};

template<>
struct RunFizzBuzz<0> // Terminate the recusion.
{
   typedef vector<int_<0> > ret;
};

int main()
{
   typedef RunFizzBuzz<100>::ret::compilation_error_here res;
}
```

```
\Main.cpp(36) : error C2039: 'compilation_error_here' : is not a member of
 'boost::mpl::vector101 <SNIP long argument list>'
    with
    [
          T0=boost::mpl::int_<0>,
          T1=boost::mpl::vector<boost::mpl::int_<1>>,
          T2=boost::mpl::vector<boost::mpl::int_<2>>,
          T3=Fizz,
          T4=boost::mpl::vector<boost::mpl::int_<4>>,
          T5=Buzz,
          T6=Fizz,
          T7=boost::mpl::vector<boost::mpl::int_<7>>,
          T8=boost::mpl::vector<boost::mpl::int_<8>>,
          T9=Fizz,
          T10=Buzz,
          T11=boost::mpl::vector<boost::mpl::int_<11>>,
          T12=Fizz,
          T13=boost::mpl::vector<boost::mpl::int_<13>>,
          T14=boost::mpl::vector<boost::mpl::int_<14>>,
          T15=FizzBuzz,
          <SNIP of elements 16 - 95>
          T96=Fizz,
          T97=boost::mpl::vector<boost::mpl::int_<97>>,
          T98=boost::mpl::vector<boost::mpl::int_<98>>,
          T99=Fizz,
          T100=Buzz
    ]
```

http://www.adampetersen.se/articles/fizzbuzz.htm

```
...
template <
        template<typename,typename,typename> class RendererT,
        typename CameraT,
        typename ObjectsT,
        typename LightsT
> struct scene {
        template <unsigned int x, unsigned int width, unsigned int y, unsigned int height> struct kernel {
        public:
                typedef scalar::sub<
                        scalar::div<scalar::from_int<x+config::virtual_x>, scalar::from_int<config::virtual_width>>,
                        scalar::c0_5
                > u;
                typedef scalar::neg<scalar::sub<
                        scalar::div<scalar::from_int<y+config::virtual_y>, scalar::from_int<config::virtual_height>>,
                        scalar::c0_5
                >> v;

                typedef typename CameraT::template generate<u,v>::type ray;

                typedef typename RendererT<
                        CameraT, ObjectsT, LightsT
                >::template raytrace<ray,15> raytrace;
        public:
                typedef typename color::rgbf_to_rgb<typename raytrace::color> color;
        };
};
...
```

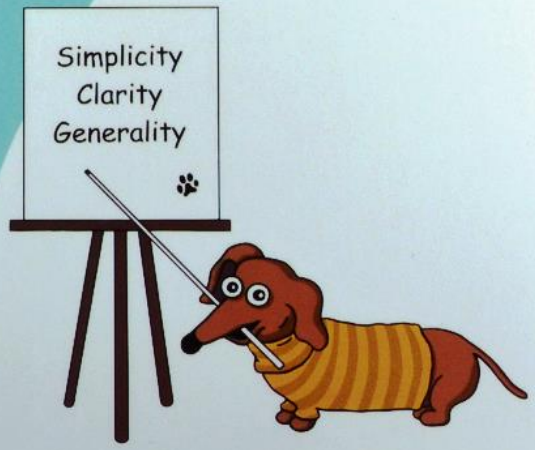
https://github.com/phresnel/metatrace

```
enum class fizzbuzzed
{
    fizz     = -2,
    buzz     = -1,
    fizzbuzz = 0,
    first    = 1,
    last     = 100
};

constexpr fizzbuzzed fizzbuzz(int n)
{
    return
        n % 15 == 0 ? fizzbuzzed::fizzbuzz :
        n % 3  == 0 ? fizzbuzzed::fizz :
        n % 5  == 0 ? fizzbuzzed::buzz :
                      fizzbuzzed(n);
}
```

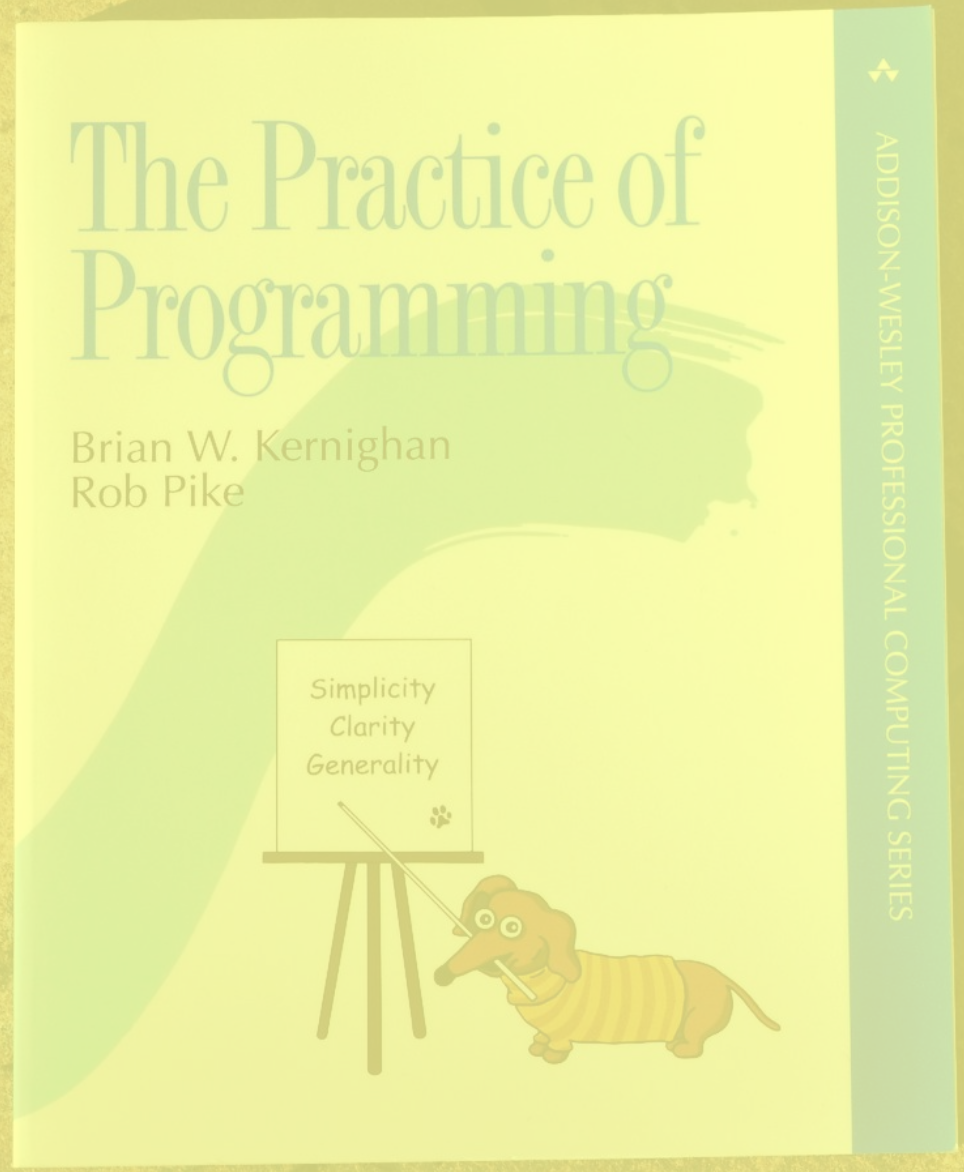
```c
#include <stdio.h>

/* printd:  print n in decimal */
void printd(int n)
{
    if (n < 0) {
        putchar('-');
        n = -n;
    }
    if (n / 10)
        printd(n / 10);
    putchar(n % 10 + '0');
}
```

The Practice of Programming

Brian W. Kernighan
Rob Pike



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES




```c
/* grep: search for regexp in file */
int grep(char *regexp, FILE *f, char *name)
{
    int n, nmatch;
    char buf[BUFSIZ];

    nmatch = 0;
    while (fgets(buf, sizeof buf, f) != NULL) {
        n = strlen(buf);
        if (n > 0 && buf[n-1] == '\n')
            buf[n-1] = '\0';
        if (match(regexp, buf)) {
            nmatch++;
            if (name != NULL)
                printf("%s:", name);
            printf("%s\n", buf);
        }
    }
    return nmatch;
}

/* matchhere: search for regexp at beginning of text */
int matchhere(char *regexp, char *text)
{
    if (regexp[0] == '\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}

/* match: search for regexp anywhere in text */
int match(char *regexp, char *text)
{
    if (regexp[0] == '^')
        return matchhere(regexp+1, text);
    do {        /* must look even if string is empty */
        if (matchhere(regexp, text))
            return 1;
    } while (*text++ != '\0');
    return 0;
}

/* matchstar: search for c*regexp at beginning of text */
int matchstar(int c, char *regexp, char *text)
{
    do {        /* a * matches zero or more instances */
        if (matchhere(regexp, text))
            return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}
```

```
int match(char *regexp, char *text)
{
    if (regexp[0] == '^')
        return matchhere(regexp+1, text);
    do {   /* must look even if string is empty */
        if (matchhere(regexp, text))
            return 1;
    } while (*text++ != '\0');
    return 0;
}

int matchhere(char *regexp, char *text)
{
    if (regexp[0] == '\0')
        return 1;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return 0;
}

int matchstar(int c, char *regexp, char *text)
{
    do {   /* a * matches zero or more instances */
        if (matchhere(regexp, text))
            return 1;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return 0;
}
```

```
bool match(const char *regexp, const char *text)
{
    if (regexp[0] == '^')
        return matchhere(regexp+1, text);
    do {
        if (matchhere(regexp, text))
            return true;
    } while (*text++ != '\0');
    return false;
}

bool matchhere(const char *regexp, const char *text)
{
    if (regexp[0] == '\0')
        return true;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text!='\0' && (regexp[0]=='.' || regexp[0]==*text))
        return matchhere(regexp+1, text+1);
    return false;
}

bool matchstar(char c, const char *regexp, const char *text)
{
    do {
        if (matchhere(regexp, text))
            return true;
    } while (*text != '\0' && (*text++ == c || c == '.'));
    return false;
}
```

```
bool match(const char *regexp, const char *text)
{
    if (regexp[0] == '^')
        return matchhere(regexp+1, text);
    do
    {
        if (matchhere(regexp, text))
            return true;
    }
    while (*text++ != '\0');
    return false;
}

bool matchhere(const char *regexp, const char *text)
{
    if (regexp[0] == '\0')
        return true;
    if (regexp[1] == '*')
        return matchstar(regexp[0], regexp+2, text);
    if (regexp[0] == '$' && regexp[1] == '\0')
        return *text == '\0';
    if (*text != '\0' && (regexp[0] == '.' || regexp[0] == *text))
        return matchhere(regexp+1, text+1);
    return false;
}

bool matchstar(char c, const char *regexp, const char *text)
{
    do
    {
        if (matchhere(regexp, text))
            return true;
    }
    while (*text != '\0' && (*text++ == c || c == '.'));
    return false;
}
```

```
bool match(const char *regexp, const char *text)
{
    if (head(regexp) == '^')
        return matchhere(tail(regexp), text);
    for (;; text = tail(text))
    {
        if (matchhere(regexp, text))
            return true;
        if (head(text) == nil)
            break;
    }
    return false;
}

bool matchhere(const char *regexp, const char *text)
{
    if (head(regexp) == nil)
        return true;
    if (head(tail(regexp)) == '*')
        return matchstar(head(regexp), tail(tail(regexp)), text);
    if (head(regexp) == '$' && head(tail(regexp)) == nil)
        return head(text) == nil;
    if (head(text) != nil && (head(regexp) == '.' || head(regexp) == head(text)))
        return matchhere(tail(regexp), tail(text));
    return false;
}

bool matchstar(char c, const char *regexp, const char *text)
{
    for(;; text = tail(text))
    {
        if (matchhere(regexp, text))
            return true;
        if (head(text) == nil || (head(text) != c && c != '.'))
            break;
    }
    return false;
}
```

```
int atexit(void (*func)(void));
```

```
void qsort(
     void *base,
     size_t nmemb, size_t size,
     int (*compar)(
          const void *, const void *));
```

```
void (*signal(
    int sig, void (*func)(int)))(int);
```

Use procedure arguments to provide flexibility in an interface.

This technique can greatly simplify an interface, eliminating a jumble of parameters that amount to a small programming language.

Butler W Lampson
"Hints for Computer System Design"

```
public class HeatingSystem
{
    public void turnOn() ...
    public void turnOff() ...
    ...
}
```

```
public class Timer
{
    public Timer(TimeOfDay toExpire, Runnable toDo) ...
    public void run() ...
    public void cancel() ...
    ...
}
```

```
Timer on =
    new Timer(
        timeToTurnOn,
        new Runnable()
        {
            public void run()
            {
                heatingSystem.turnOn();
            }
        });
Timer off =
    new Timer(
        timeToTurnOff,
        new Runnable()
        {
            public void run()
            {
                heatingSystem.turnOff();
            }
        });
```

```
class Timer
{
public:
    Timer(TimeOfDay toExpire, function<void()> toDo);
    void Run();
    void Cancel();
    ...
};
```

```
Timer on(
    timeOn,
    bind(&HeatingSystem::TurnOn, &heatingSystem));
Timer off(
    timeOff,
    bind(&HeatingSystem::TurnOff, &heatingSystem));
```

```
public class Timer
{
    public Timer(TimeOfDay toExpire, Action toDo) ...
    public void Run() ...
    public void Cancel() ...
    ...
}
```

```
Timer on = new Timer(timeOn, heatingSystem.TurnOn);
Timer off = new Timer(timeOff, heatingSystem.TurnOff);
```

```
Timer on = new Timer(timeOn, heatingSystem::turnOn);
Timer off = new Timer(timeOff, heatingSystem::turnOff);
```

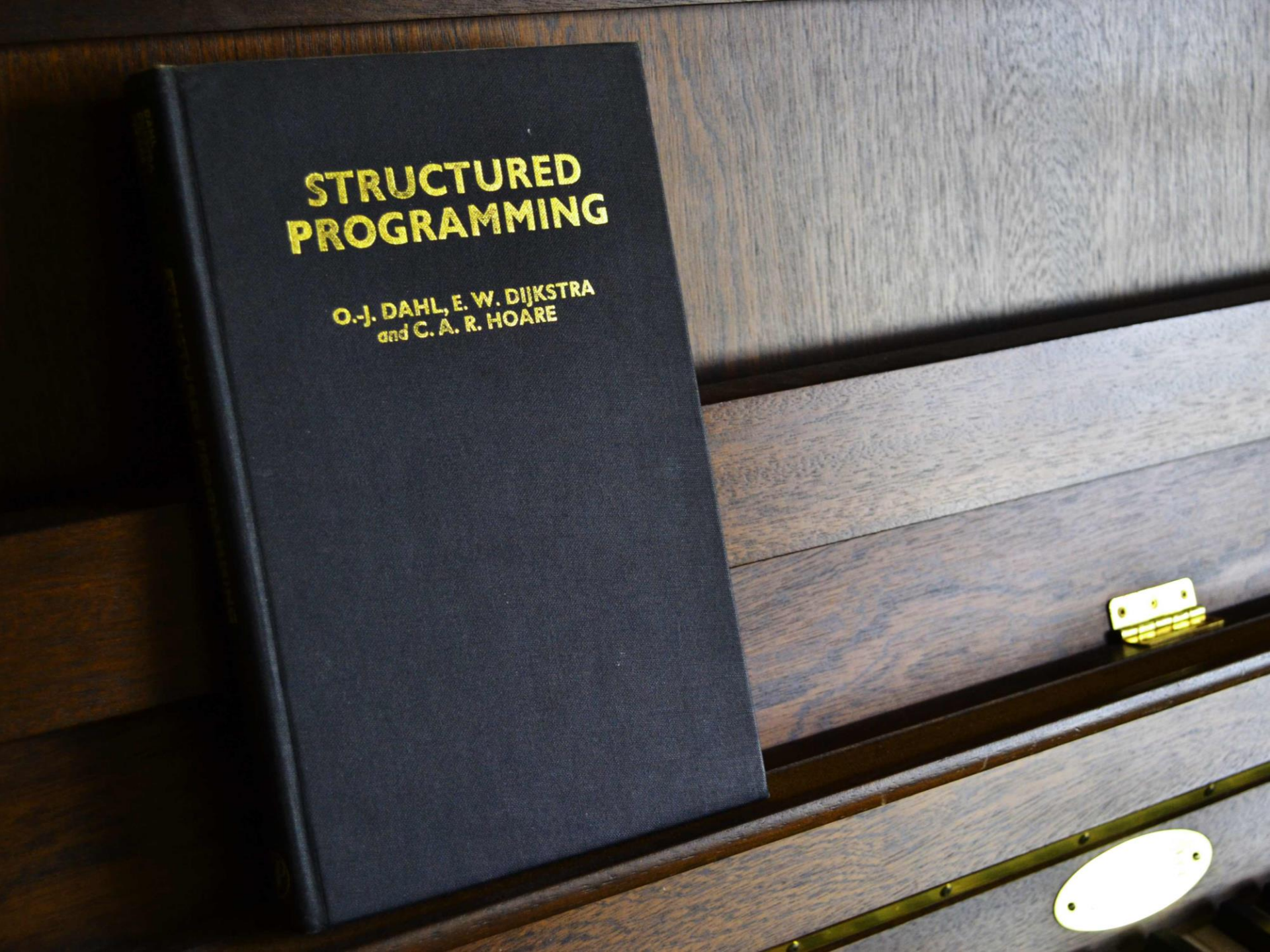
```
Timer on =
    new Timer(timeOn, () => heatingSystem.TurnOn());
Timer off =
    new Timer(timeOff, () => heatingSystem.TurnOff());
```

```
Timer on =
    new Timer(timeOn, () -> heatingSystem.turnOn());
Timer off =
    new Timer(timeOff, () -> heatingSystem.turnOff());
```

```
Timer on(
    timeOn, [&]() { heatingSystem.TurnOn(); });
Timer off(
    timeOff, [&]() { heatingSystem.TurnOff(); });
```

Lambda-calculus was the first object-oriented language (1967)

William Cook, "On Understanding Data Abstraction, Revisited"


STRUCTURED
PROGRAMMING
O.-J. DAHL, E. W. DIJKSTRA
and C. A. R. HOARE

One of the most powerful mechanisms for program structuring [...] is the block and procedure concept. [...]

A procedure which is capable of giving rise to block instances which survive its call will be known as a class; and the instances will be known as objects of that class. [...]

A call of a class generates a new object of that class.

Ole-Johan Dahl and C A R Hoare
"Hierarchical Program Structures"

intension, *n. (Logic)*

- the set of characteristics or properties by which the referent or referents of a given expression is determined; the sense of an expression that determines its reference in every possible world, as opposed to its actual reference. For example, the intension of *prime number* may be *having non-trivial integral factors*, whereas its **extension** would be the set {2, 3, 5, 7, ...}.

E J Borowski and J M Borwein
Dictionary of Mathematics

A list comprehension is a syntactic construct available in some programming languages for creating a list based on existing lists. It follows the form of the mathematical *set-builder notation* (*set comprehension*) as distinct from the use of map and filter functions.

http://en.wikipedia.org/wiki/List_comprehension

$$\{ 2 \cdot x \mid x \in \mathbb{N}, x > 0 \}$$

```
(2 * x for x in count() if x > 0)
```

$$\{ x \mid x \in \mathbb{N}, x > 0 \wedge x \bmod 2 = 0 \}$$

```
(x for x in count() if x > 0 and x % 2 == 0)
```

```
fizzes
buzzes
words
numbers
choice
fizzbuzz
```

```
fizzes   = cycle([""] * 2 + ["Fizz"])
buzzes   = cycle([""] * 4 + ["Buzz"])
words    = map(add, fizzes, buzzes)
numbers  = map(str, count(1))
choice   = max
fizzbuzz = map(choice, words, numbers)
islice(fizzbuzz, 100)
```

```
fizzes   = cycle([""] * 2 + ["Fizz"])
buzzes   = cycle([""] * 4 + ["Buzz"])
words    = map(add, fizzes, buzzes)
numbers  = map(str, count(1))
choice   = max
fizzbuzz = map(choice, words, numbers)
list(islice(fizzbuzz, 100))
```


HOLY GOD
WILL BRING
JUDGMENT
DAY ON
MAY 21, 2011
CRY MIGHTILY UNTO
GOD FOR MERCY SEE
PSALMS - 51:
JONAH - 3:
DELICATESSEN
Exit only


Richard A. Lanham
Adam Jacot de Boinod
I NEVER KNEW THERE WAS A WORD FOR IT
'Very funny'
Independent on Sunday
STEPHEN FRY
Michael Quinion
Joie de vivre n.
COMPILED BY ADRI
WORDSWORTH REFERENCE
The Wordsworth Book of Intriguing Words
REFERENCE
English mology
T. F. HOAD
word origins
the language report
A COMPREHENSIV WORD-FINDING DICTIONARY
Chambers
BILL BRYSON
TROUBLESOME WORDS
'Combines the virtues of a first-class work of reference with the pleasure of a good read' The Times
FULLY REVISED AND UPDATED
COLLINS REFERENCE DICTIONARY
MATHEMATICS
E. J. BOROWSKI AND J. M. BORWEIN
Oxford
Samuel Johnson's
LOCK
LONG WORDS
ER ME
REPENT! THE PREPOSITION IS NEAR!
JEFFREY KACIRK
AUTHOR OF Forgotten English
/ WordFriday

paraskevidekatriaphobia, *noun*

- The superstitious fear of Friday 13th.

```
struct tm next_friday_13th(struct tm when)
{
    enum { daily_secs = 24 * 60 * 60 };
    time_t seconds =
        mktime(&when) +
        (next.tm_mday == 13 ? daily_secs : 0);
    do
    {
        seconds += daily_secs;
        when = *localtime(&seconds);
    }
    while(when.tm_mday != 13 || when.tm_wday != 5);
    return when;
}
```

```
std::find_if(
    ++begin, day_iterator(),
    [](const std::tm & day)
    {
        return day.tm_mday == 13 && day.tm_wday == 5;
    });
```

```
var friday13ths =
    from day in Days.After(start)
    where day.Day == 13
    where day.DayOfWeek == DayOfWeek.Friday
    select day;

foreach(var irrationalBelief in friday13ths)
{
    ...
}
```

Concatenative programming is so called because **it uses function *composition* instead of function *application***—a non-concatenative language is thus called *applicative*.

Jon Purdy
http://evincarofautumn.blogspot.in/2012/02/why-concatenative-programming-matters.html

$$f(g(h(x)))$$

$$(f \circ g \circ h)(x)$$

$$x\ h\ g\ f$$

$$h \mid g \mid f$$

This is the basic reason Unix pipes are so powerful: they form a rudimentary string-based concatenative programming language.

Jon Purdy
http://evincarofautumn.blogspot.in/2012/02/why-concatenative-programming-matters.html

Summary--what's most important.

To put my strongest concerns in a nutshell:

1. We should have some ways of coupling programs like garden hose--screw in another segment when it becomes when it becomes necessary to massage data in another way. This is the way of IO also.

2. Our loader should be able to do link-loading and controlled establishment.

3. Our library filing scheme should allow for rather general indexing, responsibility, generations, data path switching.

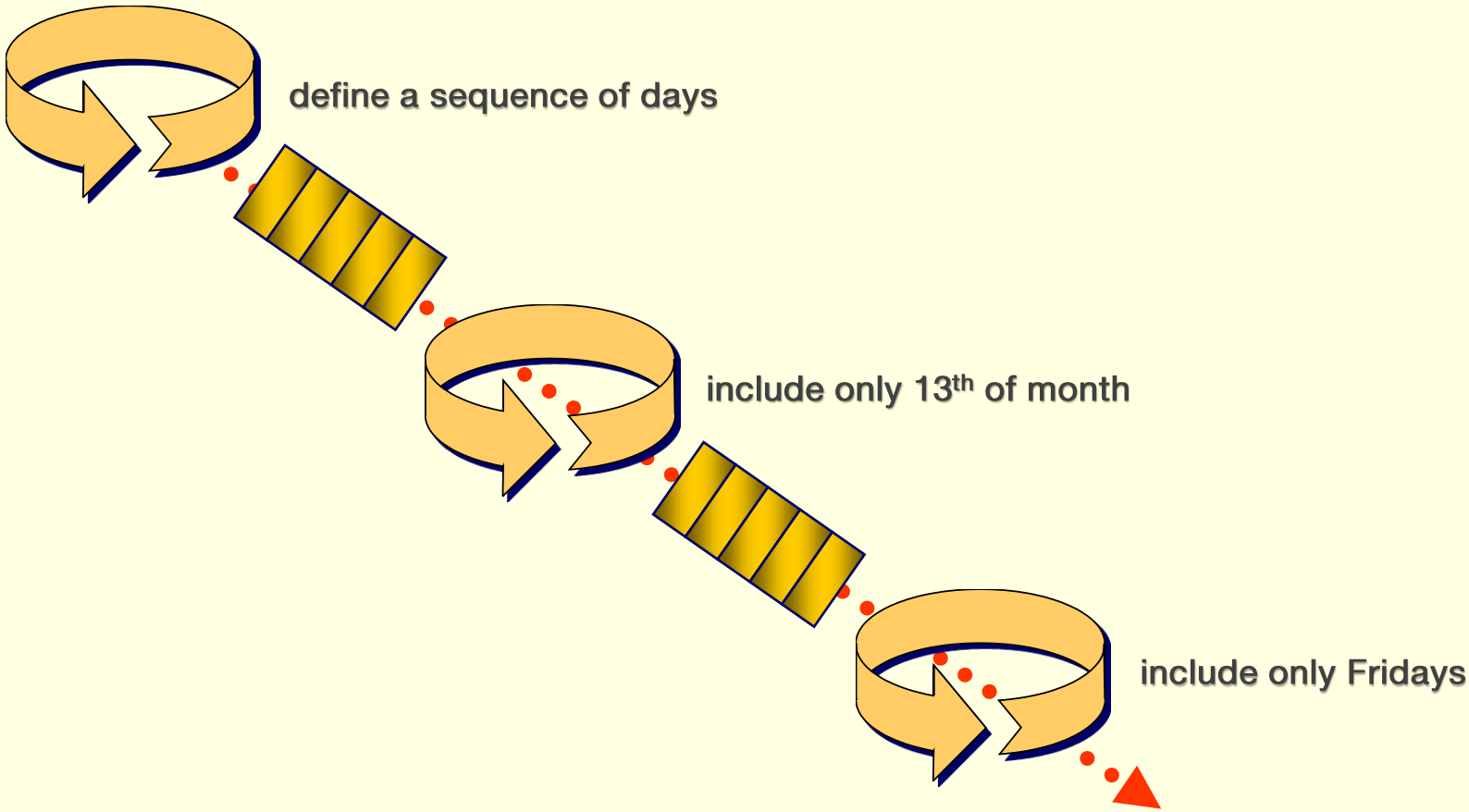
4. It should be possible to get private system components (all routines are sytem components) for buggering around with.

M. D. McIlroy
Oct. 11, 1964

```
(1..100) |
  %{
    $fizzed = if($_ % 3 -eq 0) {"Fizz"}
    $buzzed = if($_ % 5 -eq 0) {"Buzz"}
    $fizzbuzzed = $fizzed + $buzzed
    if($fizzbuzzed) {$fizzbuzzed} else {$_}
  }
```

Pipes and Filters

Divide the application's task into several self-contained data processing steps and connect these steps to a data processing pipeline via intermediate data buffers.


define a sequence of days
include only 13th of month
include only Fridays

```
function NextFriday13th($from) {
    [DateTime[]] $friday13ths =
        (1..500) |
        %{ $from.AddDays($_) } |
        ?{ $_.Day -eq 13} |
        ?{ $_.DayOfWeek -eq [DayOfWeek]::Friday }
    return $friday13ths[0]
}
```

Go with the flow.

Queens of the Stone Age