code {poems}

code {poems}
is a project by Ishac Bertran

Code editors:
David Gauthier, Jamie Allen,
Joshua Noble, Marcin Ignac

First Edition: September 2012, 100 copies
Second Edition: October 2012, 300 copies

Published by Ishac Bertran
Printed in Barcelona

code-poems.com

for ( )

FOREWORD

I sit in a room
In this room are images I cannot look at
Objects and people I cannot recognise
Codes I will never decipher
Books I will never read
I can see, I can hear
I have my own language

Poetry and computer code come out of language. Many forms of poetry can be thought of as code; sets of encrypted verses, to be read and reread, interpreted, "compiled" in the mind. The poet has always had liberty to play in this way because we are (at least) twice born: once into this world, and once into language. We know languages intimately, and are each virtuosic in our own ways, at the summoning and conjuring that they allow. *code {poems}*, as a book, is an experiment in the potential of this same intimacy, but in the realm of computer software languages. It is an investigation of the explosion of artistic forms and imaginative worlds that result from our close communion with simple sym-

bols and their arrangement on the printed page.

Computer programmers, "coders," can manipulate the material of code in the same way a poet makes present the material of language, albeit with a more esoteric, mathematical and restricted set of linguistic tools. The coder's hieroglyphics often include English-language phrasings, but these words take on skewed and concentrated meanings. In computer code, constructions like the "for-loop" and the "if-then-else," are similar to their English-language connotations, but they adopt new efficacies, new import, new powers to control the finally physical systems of hardware. And so, the facility that contemporary writers-of-code have with their symbolic systems is a different one, but no less virtuosic, and for some no less intuitive (if not more). There are those that dream in FORTRAN, and somewhere at this very moment, an impassioned conversation is taking place in Python. Long-term commitment and time spent creating software changes a person's thought patterns. A lifetime spent authoring software can change the way a mind interacts with the world. As with any expressive form, software languages change the way people think, and this has had mass-effects on culture as a whole. Despite substantial formal constraints, and intrinsic standardisation, the programming style of an author in a given language is ever present in their code. As with more "human languages," (English, German, French, etc.) machinic languages are never merely indicative or functional, but a written condensation of the mood, the personality, and the world-view of its author. A deeply hierarchical software design could indicate a cultural and political context in which a given software-hardware architecture evolves. There are seasoned readers who can differentiate the austere code of an Eastern European

collaborator from the verbose, accessible style of the Silicon Valley coder. Even within the strict rules and regulations of software, style, manner and method are distinctive, varied and personal.

The sympathies that coders extend to their more organic readership (people) are obvious, as they are usually more like what we all read daily. The use of the human-readable "comment" facility of computer languages is something of a sign of generosity toward future readers or users of that code. With all computer code though, there *might* be an intention toward a secondary audience of human readers, but there is *always* deference toward the primary electronic audience. The author's generosity, or lack thereof, toward human audiences most frequently extends in the other direction — toward the computer itself. Does the writer deal elegantly with RAM memory use, or allow the program to leave messy traces of itself all over the motherboard? Does the author stick to conventions of interfaces and interaction that the computer's operating system will "like"? And so, with a line in a text file, the author of a valuable snippet of code speaks directly to function, material action and agency (in the computer, routing electrons to-and-fro). But here, removed from prying gaze of the linker, compiler and CPU — written into a physical book like the one you now hold in your hands — code enters culture, history and imagination. It speaks uncharacteristically, and directly, to people: human readers, and our worlds.

An apocryphal story most of us have come to believe reports that the Inuit have a larger number of words for "snow" than do Anglo-American English language cultures. As a result, by accounts, these indigenous peoples' are able to understand and experience snow in a fantastically expansive way.

However contrived, the story supports our most human intuition that languages frame and effect the realities people are able to experience. The languages we use, and how we use them, seem to us a direct reflection of the richness of our lives. But language puts us both in relation to, and at distance from this life, our worlds. We describe, we create and destroy meaning, we sympathise, we embellish and beautify, we love. All the while we strain and bloat our expression in seemingly vain attempts to shape, transmit, and pay witness through words. Always trying to hit upon a perfect formulation, grappling toward descriptions of thought and feeling, as if scaling an impossibly high mountain of language (even as I type these words onto a keyboard). In this swing, between the nearly unthinkable limitlessness of language and our often frustrated expressions, lies a precarious power — used in poetry to surprise, delight and challenge. If our house of language is also a prison, poetry is an escape tunnel. The *code {poems}* in this book operate in this same way, but with constraints that are much more strict and (in contemporary computing culture) with capacities to actuate language that are in many senses even greater.

The idea of a *code {poem}* could be interpreted many ways. Most literally, we might imagine a computer writing code for us, using its mastery of rule-based permutation and iteration to generate a selection of words: infinite prose, or "a new poem every time." History has witnessed a slew of collaborations between literature and computing that operate along these lines (from computer-generated haikus to nonlinear hypertext-novels). Likewise, *code {poem}* might allude to a poem in which an encrypted message is embedded, as in the "poem codes" used as keys to decipher messages during the Second World

War (as a most famous example, Leo Marks' "The Life That I Have"). The approach taken in the collected poems included in this book, instead, is to present works that extract from the form and subject matter of computer languages *themselves*, something of their expressive power.

code {poems} presents a set of solicited works that brilliantly reveal the inner and outer workings, creative potential, and individual styles of both a particular computer program and its author. They are written by software engineers, artists and other coders, asked to "explore the potential of code to communicate at the level of poetry." (collected online at code-poems.com) The project allowed for online, public submissions from code-writers in response to the notion of a poem, written in a software language, that is semantically valid (i.e.: it would or could compile to be run on a computer). With these minimal rules, the project presents a cross-compilation of these poetic attempts themselves, each author free to interpret the idea of what *code {poetry}* might be. This collection of solicited works brilliantly reveals the inner workings, constitutive elements, and styles of both a particular software and its authors. There are poems here that specifically address classical themes like love and romance. There are others that reveal a tendency towards the kind of absolute and original control (god-complex?) that sometimes comes of hours spent building entire universes inside a machine. Still others seem to talk back to the code itself, challenging its expressivity and potential as poetry. A select few others allow world events and topical themes to emerge.

In everyday writing and speech, language itself is barely noticeable. What it is you are actually reading here and now are simple marks on a page,

disappearing right before your eyes as you read. Through its own supremacy and ubiquity, language is elusive, difficult to perceive. The words we pass our eyes over are like stealthy little cryptograms — fleeting secret messages, talking back at us in codes. It is here that poetry intervenes. Poetry is language speaking for itself, and this is no different with computer code. Those of us used to reading through code snippets and '.c' files barely notice its structure, how the syntax operates, or the oftentimes complex beauty or simple satisfaction encapsulated by each '}' or ';'. One promise of *code {poems}* is to develop new perspectives on the computer languages they feature, showing us the lyrical beauty of these structured protocols. A *code {poem}* produces for us, anew, the materiality of computer programming languages, as instructions to the machine and right there on the page.

When things get truly complex, as they may indeed be getting, the distinction between us, our tools, and the things that can be made with them begins to dissolve; the information-scapes we navigate daily are filled with media that are intertwined with an abundance of messages and identities. The signified works backward to erase the signifier; words-themselves vanish the moment they are pushed from the page or the tongue. Through *code {poems}*, we are compelled toward a re-appearance of language that illuminates how meaning is created, for us and for our machines. But only we can truly read these human encodings, these poems in code.

Jamie Allen

ABOUT THE PROJECT

Poetry is a form of literary art in which language is used for its aesthetic and evocative qualities. It allows for a multiplicity of interpretations and therefore resonates differently with each reader.

Computer code is a set of languages used to communicate with and between computers. It has its own rules (syntax) and meanings (semantics). Like literature writers or poets, coders also have their own style that includes strategies for optimizing the code as read by a computer, and facilitating its understanding through visual organization and comments for other coders.

Code can speak literature, logic, maths. It contains different layers of abstraction and it links them to the physical world of processors and memory chips. All these resources can contribute to expanding the boundaries of contemporary poetry; using code as a properly new language. Code can speak about life and death, love and hate. Code that is meant to be read, not run.

In order to explore the potential of code to communicate at the level of poetry, a call for submissions was open between the February 22nd and May 31st of 2012. The rules for submitting code poems were simple: (1) the poem having a maximum size of 0,5 KB, and (2) it required to compile.

A total of 190 poems were submitted from 30 different countries. The code editors that collaborated on the project made the selection to be printed in this book, attempting to represent the variety and creativity of the submissions, as well as different approaches to code poetry.

The poems in this book do not necessarily need to be read in any particular order. That said, the linear flow of the book has been sorted with an intention to provide a balanced and enjoyable flow to the reader.

ACKNOWLEDGEMENTS

Thanks to all the code poets that contributed to this project:

Aaron Broder, Alejandro Corredor, Álvaro Matías Wong Díaz, Andrew Couch, Andrew Parker, Antonio Moujadami, Atanas Bozdarov, Attila Palfalusi, Aymeric Mansoux, Bacchus Beale, Ben Englisch, Brad Sorensen, Bram De Buyser, Bruno Herbelin, Carrie Padian, Chris Adams, Chris Boucher, Cosima Dipalma, Dan Brown, Dane Hillard, Daniel Bezerra, Dave McKellar, Dave Mezee, David Berry, David Cantillon, David Devanny, David Homes, David Sjunnesson, Dean M Kukol, Dom Slatford, Ed Schenk, Elena Machkasova, Erik Knechtel, Giulio Presazzi, Gorenje Smack, Guilherme Kerr, Iris Dunkle, Irtaza Barlas, Izzy Edwards, Jake Forsberg, James Grant, Jason Kopylec, Jason Rowland, Jasper Speicher, Jeffrey Knight, Jennifer Mace, Jerome Saint-Clair, Jesse Pascoe, Joaquim d'Souza, John Dale, John McGuiness, John Saylor, Jolene Dunne, Jon Bounds, Jon Coe, Jonny Plackett, Jose Portelo, Josh Fongheiser, Joshua Reisenauer, Jot Kali, Ken Hubbell, Kenny Brown, Lans Nelson, Lutalo Joseph, Magda

Arques, Marc van der Holst, Marco Triverio, Marcus Ross, Mario Sangiorgio, Mark Whybird, Mary Alexandra Agner, Matias Chomicki, Matt Painter, Matthew Painter, Matthew Perkins, Matthew Ward, Michael Cheung, Michael Fall, Mikey Hogarth, Nancy Mauro-Flude, Nataliya Petkova, Nemesis Fixx, Nicholas Starke, Nick Daly, Pall Thayer, Paul Illingworth, Peter Schonefeld, Petroula Sepeta, Rafael Romero, Ramsey Nasser, Rena Mosteirin, Ricardo Sismeiro, Richard Fletcher, Richard Littauer, Roger Donat, Ruggero Castagnola, Ryan Christiansen, Ryan Kabir, Shani Naeema, Shawn Lawson, Signe Breum, Soon Van, Suhail Thakur, Sylke Boyd, Terek Ertman, Thibault Autheman, Thomas Braun, Thomas Pellegrini, Tobby Cheruthuruthil, Ubaldo Pescatore, V Nels, Vilson Vieira, Viviana Alvarez Chomón, William Dupré, William Linville, Wolf Herrera, Xtine Burrough, Yann van der Cruyssen and Yves Daoust.

Thanks to Jamie Allen, Joshua Noble, David Gauthier and Marcin Ignac for their enthusiasm and dedication to the project.

And thanks to Elena Gianni, John Lynch, Marco Triverio, Hari Harikrishnan and Harsha Vardhan for great conversations about code poetry.

Ishac Bertran

EDITION NOTE

The code {poems} in this book are set in Inconsolata, a monospace typeface designed by Raph Levien in 2005, inspired by humanist sans fonts. The rest of the text is set in Bookman Old Face, a serif typeface designed by Alexander Phemister in 1858.

The book is printed by Impremta Badia, founded in 1888 in Barcelona, in black onto 100gsm Soporset paper.

```
String silence="                    ";
String idea="This is'nt p0etry.";
String draft;
String[]poem=new String[idea.length()];
void setup(){
 draft=idea;
 Write();
 ReThink();
}
void draw(){
ReWrite();
}
void Write(){
println (draft);
}
void ReThink(){
for(int decomp=0;decomp<draft.length();decomp++)
{poem[decomp]=draft.substring(decomp,decomp+1);}
}
void ReWrite(){
 byte seek=byte(random(0, poem.length));
 poem[seek]=" ";
 String poetry=join(poem,"");
 println(poetry);
 if(poetry.equals(silence)){
 println("."); noLoop();}
}
```

Alejandro Corredor
// Processing

UNHANDLED LOVE

```
class love {};

void main()
{
    throw love();
}
```

Daniel Bezerra
// C++

BODY

```
<html>
<body>
<!-Something->this<!-life-><!-has-><!-taught-><!-
me-><br>
<!-->is <!-that->the<!-perfect-><!-body-><!-is-
><!-a myth-><br>
<!-and-><!-there's no-> secret<!-shortcut-><!-to-
>:<!-happiness-><br>
<!-and->nobody<!-is-><!-always-><!-right-><br>
<!-we-><!-all-><!-have-><!-moments-><!-of-><!-
brilliance->
<!-and->fits<!-of-><!-R-><!-A-><!-G-><!-E-><br>
<!-and-><!-live-><!-our lives-><!-beautifully->
<!-in-><!-the-><!-world's-><!-imperfect-><!-
plane->
</body>
</html>
```

Carrie Padian
// HTML

RACING NUMBERS

```
#!/bin/bash

_..() { .._  ${@//[tt]?[^s]/2};}
__.() { echo -ne $@' '       ;}
one() { __. '1' && _.. $@    ;}
and() { __. '&' && _.. $@    ;}
two() { __. '2' && _.. $@    ;}
.__() { echo -ne $@\\n       ;}
..._() { .__  ${@//[ow]?[^s]/1};}
```

Gerrit Riessen
// Shell

DISFUNCTION()

```
function disfunction(){

if(weCannotStart&&weWillNotStop){
if(iCannotRun&&uCannotWalk){
if(thereIsNoWay&&thereIsNoOne){
if(uHoldYourFlag&&iHoldMyGun){
while(timePassesBy&&hopePassesOut){
while(someRemainSilent&&someOnlyShout){
while(someWaitForJesus&&someWaitForFood){
while(uChangeYouClothes&&iChangeMyMood){
try{settlingDown||openingUp;
//try
listeningIn||!actingOut;
//try
[anything,something,whatever,whoCares];}
//all that's left when you
catch(me){isAskingWhosThere;}}}}}}}}}
```

Brad Sorensen
// Javascript

IMPORT SOUL

```
# This script should save lives.

import soul

for days in len(life):
  print "happiness"
```

Richard Littauer
// Python

MARY.LAMB

```
enum Color {
    white,
    black,
    dirty
};

class Lamb {
    public:
    Color fleece;

    Lamb();
};

Lamb::Lamb(): fleece(white)
{
}

class Shepherd {
    Lamb lamb;
};

int main(int argc, char* argv[])
{
    Shepherd mary;

    goto school;

    return (0);

school:
    return (5);
}
```

James Grant
// C++

FLUENT.SHE.SHARPLY

```
namespace msp.She.Sharply
{
   public class S
   {
      S myJoy;
      S withMyChildren;
      S everSoSweetly;
      S asTheHeroine;
      S smilingWithArms;
      S laughingAtMyJokes;
      S sleepilyStretching;
      S dancingInTheKitchen;

      public S asISing(S you)
      {
         myJoy = you;
         withMyChildren =
            everSoSweetly =
               asTheHeroine =
               smilingWithArms =
               laughingAtMyJokes =
               sleepilyStretching =
               dancingInTheKitchen = myJoy;
         return everSoSweetly;
      }
   }
}
```

Matthew Perkins

// C#

UNTITLED (LOVE)

```
float intime;
//intimate the presence of the flesh
int erlace;
//for the
         void /*of mountains*/
             /*wilderness*/
setup(/*silence in the*/){background(0/*f*/);//a
shivering lake
/*when it*/
            stroke/*s your skin*/(255 /*pulsing
beats*/);}
// in this
         void /*that*/draw/*s the frequency of a
kiss*/(){
//for there is no accurate measure
/*when the trembling*/line
(/*int*/erlace/*s with my*/,50/*ar heart*/,100/
*se*/,50/*ak*/);
}
```

Nataliya Petkova

// Processing

```
#include <stdio.h>

unsigned int n = 1701998444;

int main()
{
    int i = 0;

    while(i < n) {
        putc(((n >> (i % 4) * 8) & 0xff) +
             4 * ((int) (i % 8 % 7) / 6),
             stdout);
        i++;
    }

    return n;
}
```

Marco Triverio

// C


```
int comma(int hyphen, int greater)
{
    if (greater > hyphen)
    {
        return comma(greater - hyphen, hyphen);
    }
    else if (hyphen > greater)
    {
        return comma(hyphen - greater, greater);
    }
    else
    {
        return greater, hyphen;
    }
}
```

Yves Daoust

// C

FOLLOW ME TO THE DEN OF ZEN...

```
#follow me to the den of zen...

seek='wisdom in the unknown'
seek and nil;{
   'with awe'=>'at the altar o ruby gods o
               old'.split(/your curiosity/),
   'u must'=>((1/2 and 'conquer')
   if seek+'ing enlightenment!' )}

begin
   to=seek
   seek,far=to,seek
   if 'u seek'<< 1.abs << 'truth':
      to << 'ever ascend beyond' <<
            (1/0.0).infinite?
   end; 'yo faith'
end

this = 666.times
{|sin| "brings > light than darkness" }
"readin #{this} in the bible o satan, i thought"
["i","should"].join"them"
print this and seek << 'divinity in my heart'
```

Nemessis Fixx
// Ruby

ISM | BREATH | WHO | SHE | WITH | I

```
cat roomofonesown.txt | sed 's/a[A-Z]/[a-z]/g' |
grep -oE "\b[a-z]+ism\b" | sort | uniq -c | sort
   1 despotism
   1 feminism
   1 organism
   1 scepticism
   1 symbolism
  13 criticism

cat roomofonesown.txt | grep -oE "\b[a-z]+
breath\b"
drew breath
her breath
his breath
hot breath
my breath
our breath
```

Nancy Mauro-Flude
*// *nix*

TWOFACED

```
public class TwoFaced {
    public String greet() {
        return "Hi! So great to see you!";
    }

    private String think() {
        return "Fucking bitch.";
    }
}
```

Jason Kopylec

// Java

BITS OF THE UNIVERSE

```
nil
pop!
(time ())
empty?
atom
atom  atom
atom atom atom atom atom
atom atom atom atom atom atom atom atom atom
(binding [atom [atom [atom [atom [atom]]]]])
make-hierarchy
repeat
repeatedly iterate sequence
constantly interleave
merge

replicate
parents
descendants
cycle
7000000000

true? false?
find identity
find name
symbol? number?
rational? odd?
resolve
future?
future-done?
future-cancelled?
reversible?
nil?
```

Elena Machkasova

// Clojure

LIFE IS RANDOM

```
import java.util.*;

public class Life {
    private String brain;
    private String fate;
    public static void main(String[] a) {
        Life I = new Life();
        I.live();
    }

    public void live() {
        my_destiny();
        System.out.println("I'm "+brain+" and
"+fate+".");
    }

    public void my_destiny() {
    Random r = new Random();
    String [] fates = {"Lucky", "Unlucky"};
    String [] brains = {"Smart", "Stupid"};
    fate = fates[r.nextInt(2)];
    brain = brains[r.nextInt(2)];
    }
}
```

Ubaldo Pescatore

// Java

FOR AGNES

```
for you_Agnes in `which time`;
do find /if you can/ \
\
"there is nothing, Agnes" 2> be_done;
until [[ $you = **know** ]];

do you="know `whoami` ?";
look " my Agnes ... ";

if [ -u "only could" ];
then id rejoice;

patch false hopes;   fi;
from nothing 2> nothing;

done;
done;

less $0 |\
say -v Agnes
```

Jeffrey Knight

// Shell (OSX)

SIMPLIFY

```
Module Simplicity
    Dim Simplicity
    Dim good
    Sub Main()
        Simplicity = good
        ' To gain it,
        ' Distractions must be removed
        ' And clutter eradicated. So,
        ' Simplify! Remove the garbage!
        If My.Computer.FileSystem.FileExists("C:
\globdata.ini") = True Then
My.Computer.FileSystem.DeleteFile("C:
\globdata.ini")
        Console.WriteLine("Now that you've
simplified, relax.")
    End Sub
End Module
```

Josh Fongheiser

// Visual Basic

TWO STEPS FORWARD

```
#include <stdbool.h>

int main(int argc, const char *argv[]) {
    int x = 1;
    while (true) {
        x = x << 2;
        x = x >> 1;
    }

    return 0;
}
```

Aaron Broder

// C

RISING

```
# Three.py ~ Rising

from time import sleep
from random import random

mom = open(__file__).read()

while True:
    child = open(str(random()) + '.py', 'w')
    child.write(mom)
    child.close()
    sleep(random()*10)
```

Vilson Vieira, Renato Fabbri
// Phyton

OSLO

```
Oslo =          ["hot",
                      "and",
                      "dry"]
in_May =        "for roses"
Cold =          ["but",
                      "temperatures",
                      "run higher"]
Still =         ""
Faces =         ["suppress",
                      "grained, white, screens",
                      "hardly seen"]
Absorb =        "Fascism"
The_judgement = ["most exact",
                      "boy & girl",
                      "lie flat"]
Stark =    ""
Eyes =          ["white-fimbriated",
                      "blue",
                      "white, lies"]
Red =           ""
For_the =       ["love of God",
                      "July 22 remains",
                      "Breivik's trial of the"]
People =   ""
```

David Berry
// Ruby

THE RUMOR

```
#!/usr/bin/perl
if(true){
                true unless false;
}else{
                false unless true;
}
```

Pall Thayer
// Perl

JUDGMENT

```
<SCRIPT>

judgment= function(you){

 qtn='How many things do you '

 you.guess=prompt(qtn+'guess?',0)
 you.know =prompt(qtn+'know?' ,0)

 induction='fai';deduction='suc'
 arrogance='lu' ;humility ='ce'
 prejudice='re' ;tolerance='ss'

 ignorance=induction+arrogance+prejudice
 wisdom   =deduction+humility +tolerance
 prudence =you.know-you.guess

 this.method=(prudence<1)?ignorance:wisdom

};

m=new judgment({}).method
document.write('This judgment tends to '+m)

</SCRIPT>
```

Guilherme Kerr
// Javascript

NESTING

```
<GOD>
 <universe>
  <galaxy>
   <solarsystem>
    <earth>
     <island>
      <town>
       <garden>
        <flowerbed>
         <snowdrop>
          <petal>
           <molecule>
            <proton>
             <quark>
              <GOD>
             </quark>
            </proton>
           </molecule>
          </petal>
         </snowdrop>
        </flowerbed>
       </garden>
      </town>
     </island>
    </earth>
   </solarsystem>
  </galaxy>
 </universe>
</GOD>
```

Dan Brown
// HTML

THE GAME

```
boots: "slush"
feet: "slip"
football: [sails past fingertips]

while [chuckling: "fathers"] [
    watch-the-world-turn-past: "counter-clockwise"
]

a: [ball almost caught]
```

Ryan Christiansen
// Rebol

A VOLATILE SKETCHBOOK

```
\ A Volatile Sketchbook
\ for the vaporware artist


variable page 0 cells allot
0 constant pencil
1 constant charcoal


: draw  1 + + dup 1 + swap 2drop ;
: turn  2 page +! ;


: sketch
        for
                page @ dup
                charcoal draw
                pencil draw
                turn
        next ;


: book  255 sketch ;


book
```

Aymeric Mansoux
// FORTH

N/A

```
class ic_case {

        void of_ambition() {


                while(true) {

                                        //TODO

                }

        }

}
```

Joaquim d'Souza
//Java

```
import flash.utils.Timer;

var steveTimer = new Timer( 1000 );
steveTimer.addEventListener(TimerEvent.TIMER,
stillAlive);
steveTimer.start();

function stillAlive(e)
    {
    trace("Steve tried to killed me, but I've
    lived " + ( Math.round(new Date().getTime()/
    1000) - 1317772800 ) + " seconds longer.");
    }
```

Jonny Plackett

// AS3


```
#include<iostream>
int i,j,l,r,o[80]={},t[80]={};
char c;
int main(){
 std::cout<<"enter character, then return >> ";
 std::cin>>c;
 o[39]=1;
 while(1){
   for(j=0;j<80;j++){
     if(j)     l=o[j-1]; else l=o[79];
     if(j!=79) r=o[j+1]; else r=o[0];
     if(o[j])
       if(l)   t[j]=0;   else t[j]=1;
     else
       if(l^r) t[j]=1;   else t[j]=0;
     std::cout<<((o[j]?" ":&c));
   }
   for(;--j>=0;) o[j]=t[j];
   std::cout.flush();
   usleep(50000);
 }
}
```

Shawn Lawson

// C++

VARIATIONS ON A QUINE, IN PY MINOR

```
s="n=chr(10);c=chr(34);print's='+c+s+c+n+s"
n=chr(10);c=chr(34);print's='+c+s+c+n+s
```

Rafael Romero
// Python

TIME GOES BY SLOWLY

```
Class TimeGoesbySlowly

Function ToSave(YourLife)

For Each Day As TimeGoesbySlowly In YourLife

Dim theLights As New TimeGoesbySlowly, TaketheTime

ToSave(YourLife) 'and

Return theLights 'until the

Next Day

End Function

End Class
```

Chris Boucher
// Visual Basic

VAN[ ]SH

```
int light = int(random(0,100));
int i;
char [] shadow = new char [light];
  //Evanescent electrons for constructed vacuum
  //appear.
  //A spectral presence
  //unfolds.

void setup()
{
  self(i);
  //Execute.
}

void self(int i)
{
  //The random data flow
  //draws step by step
  //a carbon copy
  //of the vector shadow
  shadow[i]= '0';
  print(shadow[i]);

  if(i<light-1)
  {
  self(i+1);
  //The transport of unmovable parts has begun.
  }
}
```

Thibault Autheman
// Processing

CESARE PAVESE'S STUBBORNESS

```
String whatHappens;
BufferedReader today;
PImage littleSweetFruits;

void setup() {
  today = createReader
  ("the_world_but_what_i_m_waiting_for.txt");
  littleSweetFruits = loadImage
  ("i_dress_them_up_with_salt.jpg");
}

void draw() {
  try
  {whatHappens = today.readLine();
  /* just things */ }

  catch
  (IOException e)
  {/* so i don't want to see */
   whatHappens = null;}}
```

Ruggero Castagnola
// Processing

AN OCEAN OF INTEGER WAVES

```
public class IntWaves
{
public static void main(String[] a)
throws Throwable
{
  int h=25;
  while(true)
  {
    java.io.PrintStream p = System.out;
    int cnt=new java.util.Random().nextInt(h);
    int i=0,j=0;
    for(;i<=cnt;i++)
    {
      for(j=0;j<i;j++){p.print(i);}
      p.println();
      Thread.sleep(h);
    }
    for(i=cnt;i>=0;i--)
    {
      for(j=0;j<i;j++){p.print(i);}
      p.println();
      Thread.sleep(h);
    }
  }
}
}
```

Matthew Ward

// Java

IMMEDIATE FUNCTION

```
(function() {
   var poem  = 'I am feeling good,\n';
      poem += 'and feel so fine - \n';
      poem += 'I just get parsed, \n';
      poem += 'a single time';
   alert(poem);
})();
```

Marcus Ross

// Javascript

HTML = HELLO TODAY MEANS LATER

```
<html>

<head>

<title>Hello Today Means Later</title>

<style type="text/css">

   you{display: block;}
   me{display: block;}

</style></head>

<body>

   <me>Oscillate between your eyes</me>
   <you>Routine of breaths</you>
   <me>Folk</me>&
   <you>Fuck</you>
   <time>Condition of latency & exposure</time>
   <you>An answer</you>
   <me>Your Hands</me>
   <time>Sailed to the drift</time>
   <you>Telepathic Seas</you>
   <me>Searching with the Waves</me>
   <time>Loop forever</time>

</body>

</html>
```

Viviana Alvarez
// HTML

BASICHELL

```
10 mem conundrum
25 lookup 30
30 goto 10
```

Wolf Herrera
// DOS

OH MINI 8-BALL

```
program OhMini8Ball;
var Quest:String;
P:array[0..2] of String=('Yes','No','Maybe');
begin Randomize;
WriteLn('What profound answer you ask of me?');
ReadLn(Quest);
if Quest<>'' then begin
WriteLn('From wonders and '+
'mysteries this is given thee');
WriteLn('The Wisdom of the Mini 8-Ball'+
' has for you to see:');
WriteLn(P[Random(3)]);
WriteLn('Come again to seek'+
' only what I may give for free.');
end else WriteLn('Only an empty answer'+
' from a question where the words are free.');end.
```

Michael Fall

// Pascal

SATURDAY, FIRST LIGHT

```
#include "early_morning_run.h"
int main(void) {  int spanish_wine = 0;
char first_beat[] = "dawn over the lake;";
int alcohol_age = 21;

char epaulette[54] =
"waves break"; char *no_fuss; // empty as yet
if (alcohol_age < 34) {
no_fuss = strcat(epaulette,

" yesterday's bottles");} spanish_wine++;
printf("%s\n",first_beat);
printf("%s\n",no_fuss);
char end_line[] = "against the concrete";

printf("%s\n",end_line);
printf("\n"); return spanish_wine;}
```

Jennifer Mace

// C

BUDDHISTLIFE

```
public class BuddhistLife {

    boolean alive = true;
    boolean enlightened = false;

    public BuddhistLife() {
        while (alive) {
            fortune();
        }
        dispose();
    }

    private void fortune() {
        if (((int)(Math.random()*365)) ==  108){
            alive = false;
        }
        if (((int)(Math.random()*3650)) == 108){
            enlightened = true;
        }
    }

    private void dispose() {
        if (!enlightened) {
            new BuddhistLife();
        }
    }

    public static void main(String[] args) {
        new BuddhistLife();
    }
}
```

Thomas Braun

// Java

BODYCLOCK

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
namespace BodyClock{
 class Program{
  static void Main(string[] args){
   BodyClock();
   Console.ReadLine();}
  public static void BodyClock(){
   int n, Day = 1, Year = 0;
   bool Life = true;
   n = 1;
   while (Life == true){
    Day += 1;
   if (Day / (365 * n) == 0){
    Year += 1;
    n += 1;
    Console.WriteLine("Happy Birthday your {0}",
Year);}
   if (Year == 78){
    Life = false;}}}}}
```

Dom Slatford

// C#

REALITY

```
if (humans!=robots) {
  reality();
}
```

David Sjunnesson
// Processing

HOW TO CHOOSE A LOVER WITH SQL

```
SELECT [Love].Name,

IIf([Brains]>[Beauty],"Theatre","Booty-Call")
   AS [Type_Of_Date],

IIf([Love].[True] Like "True","False",
   IIf([Heartbreaker] Like "True","Bad idea",
   "Sure, why not?"))
   AS [Good_For_A_Fling],

IIf([Height_In_Ft]>5.5,"Y","N")
   AS [Tall_Enough],

IIf([Trust] Like "No","No way",IIf([Love].[True]
Like "False","Nah","Absolutely!"))
   AS [Marriage_Material]

FROM ([Love] INNER JOIN Body ON [Love].Name =
Body.Name)
   INNER JOIN Mind ON Body.Name = Mind.Name;
```

David Devanny
// SQL

SLEEPINGTHROUGHLIFE

```c
#include <stdio.h>
#define LIFESPAN 81

int main () {

int age = 0;
int death = 0;
int life = 1;

while (! death) {
    age++;
    sleep(31556926);
    if (age == LIFESPAN) death = life;
}

}
```

Jot Kali
// C

DAILYGRIND

```java
import java.util.Date;

public class DailyGrind {

   public static final void main(String[] args) {

      boolean its_time_to_go_home = false;
      boolean away_the_hours = true;

      while (away_the_hours) {

         Date now = new Date();
         its_time_to_go_home = now.getHours() > 17
               && now.getMinutes() > 30;

         if (its_time_to_go_home) {
            break;
         }

         try {
            Thread.sleep(60000);
         } catch (InterruptedException e) {
            // ignore
         }
      }
   }
}
```

Paul Illingworth
// Java

FOREVER

```c
/* It fades, but it never goes away */

#include<stdio.h>

// Here I am
int i;

// I need some words
#define hey          do{printf("%s\n",
#define forever      );sleep(++i);}while(1);

int main(){

        hey
        "you will miss me"
        forever

}
```

Mario Sangiorgio
// C

SHORT TRIPS

```
◁='⊂'
▷='⊃'
△='∩'
▽='∪'

_={}
¯={'_':eval,'¯':'parseInt(Math.random()*9)',
  '__':'document.write(" '}

╪=function(-,--) {
  |=--[¯[-]++%4]
  ||=_[--[0]]
  for(¨=0;¨<¯[|];¨++) {¯._(¯.__+|+'")')}
}

_[◁]=[◁,▷,△,▽]
_[▷]=[▷,△,▽,◁]
_[△]=[△,▽,◁,▷]
_[▽]=[▽,◁,▷,△]

¯[◁]=¯._(¯.¯)
¯[▷]=¯._(¯.¯)
¯[△]=¯._(¯.¯)
¯[▽]=¯._(¯.¯)

|=△
||=_[|]

setInterval('╪(|,||)',10);
```

the55
// Javascript

ENDET

```
function you(now)
! Oh !
logical you
! already
now=now+1
if (now ==6) you=.false.
! you never
return
end

program endet
logical dream
logical night
logical you
!!
character*3 me
me='ill'

now=0
night=.true.
! I toss and turn and
do while(night)
    dream = you(now)
    night=dream
    !Oh yes, please
    write(*,*) dream, me
    ! On the sheets
    write(*,*)'  me with your',night,'ire: ', now
enddo
end
```

Sylke Boyd
// Fortran 90

OPTIMIZE ME

```
int main()
{
    int i = 0;

    i++;
    i--;

    return i;
}
```

Daniel Bezerra
// C++

INSERT IGNORANCE INTO IGNORANCE

```
  String     ignorance = "ignorance";
   class            ignorance
 {String            insertInto
 (String            ignorance
){return            ignorance;}}

void draw() {   ignorance ign0rance
;


           /************************/
/**********/ign0rance = new ignorance()/*********/
;          /************************/


println(  ign0rance.insertInto(ignorance)
);}
```

Ed Schenk
// Processing

LOVE WILL TEAR US APART

```
public class LoveWillTearUsApart {

   public static void love(String y, String m,
String l){
     if (y!=m) {
        l.split("&");
     }
     love(y, m, l);
   }

   public static void main(String[] args) {
      LoveWillTearUsApart.love("my_road",
"your_road","you&me");
   }
}
```

Jerome Saint-Clair
// Java

DESPERATE PROGRAM

```
var ious = {
  holding : [ 'brie' + 'f', { psuedo : 'secular'
},
    [ 'med', 'it', 'at', 'ions' ]
  ],
  message : "not understanding\n",
  barely : function( al ) { alert( 'when will you
awake?' ); },

  // unrecoverable errors
  stop : function( ing ) {
    for( var y = 0; y < ious[ 'holding' ][ 2
].length; y++ ) {
      var ied = confirm( ious.message + ious[
'holding' ][ 2 ][ y ] );
      if( ! ied ) { setTimeout( ious.barely, 10000
); }
    }
  }
};
```

John Saylor

// Javascript

EPISTLE

```
class LookCloser
{
  public: bool broken, purpose, ornament;
  public: LookCloser(
      bool broken,
      bool purpose,
      bool ornament)
  {
    this->broken =
      (broken ^ purpose) &&
      (purpose || ornament);
    this->purpose =
      !purpose ||
      (broken && ornament);
    this->ornament =
      (purpose ^ ornament) &&
      !(broken && ornament);
  }
};
```

Yann van der Cruyssen

// C++

```
using System;

public class PoemCode
{
   private bool dancing_within()
   {
      Boolean me = true;
      while (dancing_within())
      {
         var iables_of_light = "";
         try { int elligently_to;
         object ify_the_world_apart; }
         catch (Exception s)
         {
            int o_the_broken_parts;
            throw; int o_the_seed_of_life;
         }

         Random ashes_of = new Random();
         float ing_devices;
         short age_of;
         char acter_will_never_let_you = 'b';
      }
      return me;
   }
}
```

Álvaro Matías Wong Díaz
// C#


```
# Creation
def dstBit(mass,rot,vel):
        bMass=mass
        bRot=rot
        bVel=vel

def dstCld(mass,rot):
        mass=mass
        rot=rot

def Stir(dstBit1,dstBit2):
        CldMass=dstBit1.mass+dstBit2.mass
        CldRot=dstBit1.vel*dstBit2.vel
        return dstCld(CldMass,CldRot)

def Sprk(dstCld):return StellarObject(dstCld.mass)

def Life(planet,seed):return None

dstBit1=dstBit(8.3,5.2,-7.1)
dstBit2=dstBit(5.3,3.2,5.4)

Cld=Stir(dstBit1,dstBit2)
Planets=[]
for i in range(8):
        Planets[i]=Stir(Cld,dstBit1)

Sol=Sprk(Cld)

Life(Planets[2])
```

Kenny Brown
// Python