CSCI 334:
Principles of Programming Languages

Lecture 14: ML

Instructor: Dan Barowy

Williams

Topics

Project ideas

ML family of languages

F#

Your to-dos

1. Lab 6, **due Sunday 4/10** (partner lab)
2. Reading response, **due Wednesday 4/13**.

Announcements

- **Field trip to WCMA**, Tuesday, April 12.
  Bring your handout from our first trip.
- Please **consider being a TA** next semester
  (especially for this class!)

  Applications **due Friday, April 22**.

  https://csci.williams.edu/tatutor-application/

Announcements

Colloquium on Friday.

Friday, April 8 @ 2:35pm

Wege Hall – TCL 123

Perception and Context in Data Visualization

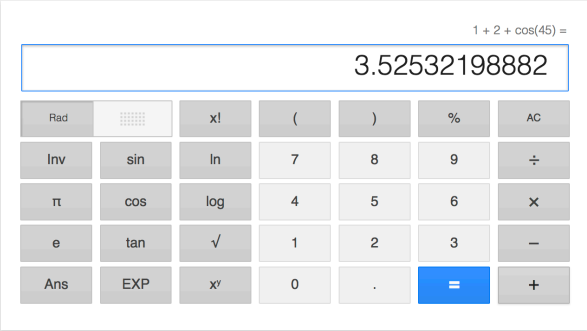
Jordan Crouser, Smith College

Visual analytics is the science of combining interactive visual interfaces and information visualization techniques with automatic algorithms to support analytical reasoning through human-computer interaction. People use visual analytics tools and techniques to synthesize information and derive insight from massive, dynamic, ambiguous, and often conflicting data… and we exploit all kinds of perceptual tricks to do it! In this talk, we'll explore concepts in decision-making, human perception, and color theory as they apply to data-driven communication. Whether you're an aspiring data scientist or you're just curious about the mechanics of how data visualization works under the hood, stop by and take your pre-attentive processing for a spin.

Inspiration for Projects

Project to be discussed at WCMA

Scientific Calculator



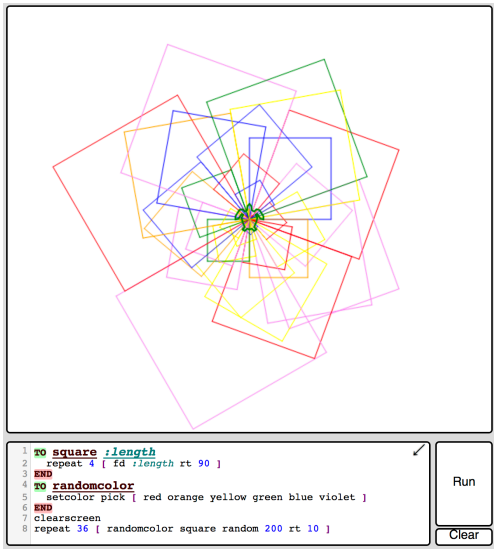
https://www.google.com/search?q=google+calculator

1 + 2 * 3 - 1

Should support:

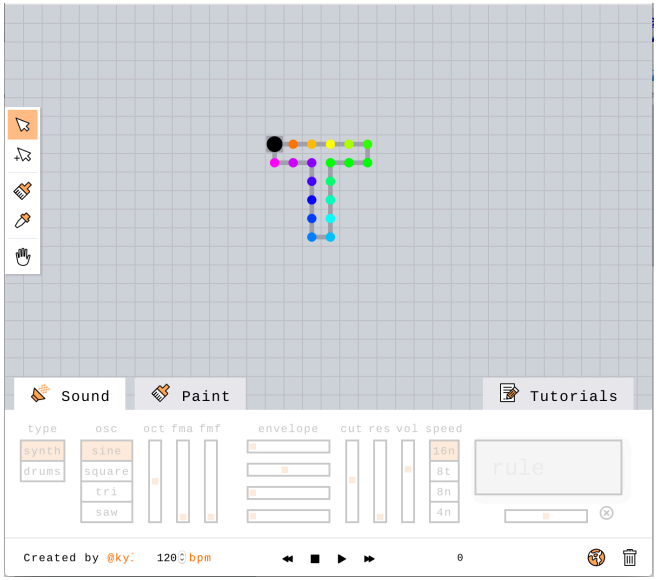
- infix arithmetic AND
- variables, or
- user-defined functions, or
- add some other interesting feature of your own design.

Logo Interpreter



https://www.calormen.com/jslogo/

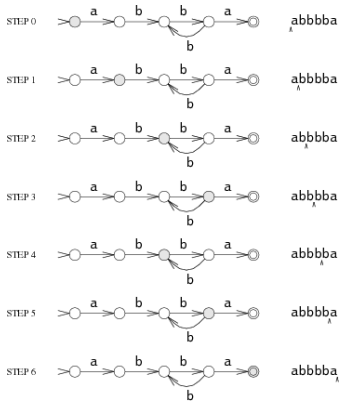
Turtle Audio



http://www.turtle.audio/

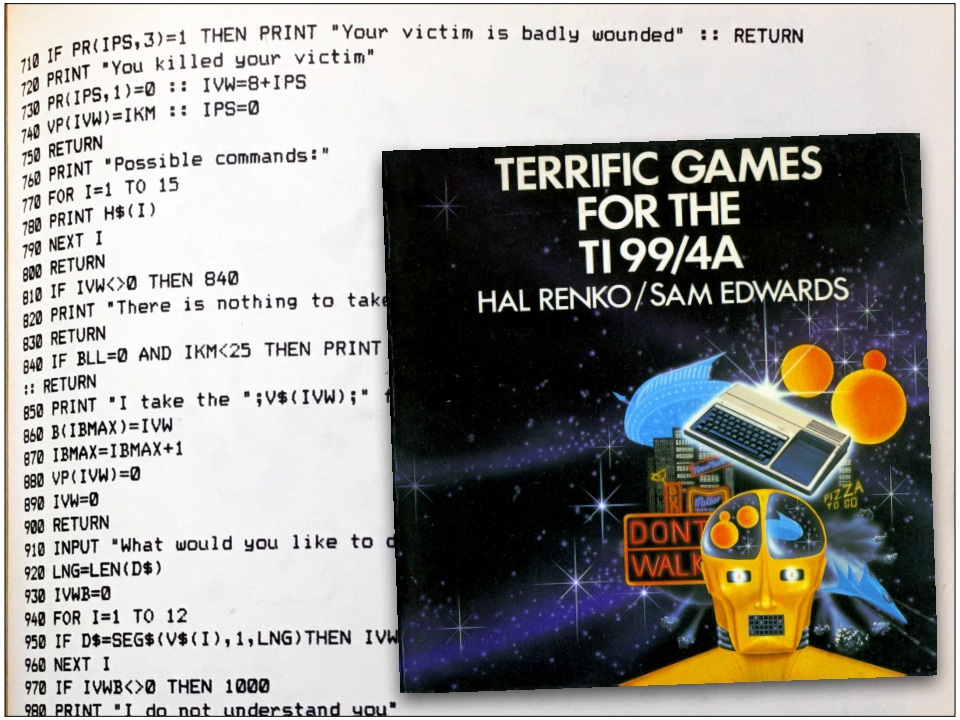
Regular Expressions

Regular Expression Matching Can Be Simple And Fast
(but is slow in Java, Perl, PHP, Python, Ruby, ...)

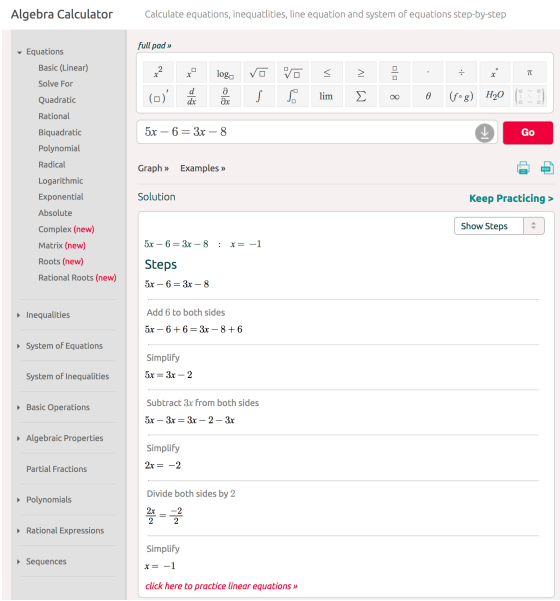


https://swtch.com/~rsc/regexp/regexp1.html

BASIC



An algebra solver



https://www.symbolab.com/solver/algebra-calculator

ChucK

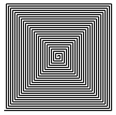
```
SinOsc ge => dac;

while( true )
{
  Math.random2f(30,1000) => ge.freq;
  .5::second => now;
}
```

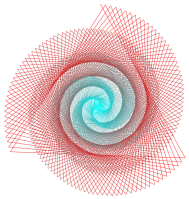
https://www.ted.com/talks/ge_wang_the_diy_orchestra_of_the_future

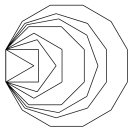
http://chuck.cs.princeton.edu/

Highlights from Previous Classes

Zig

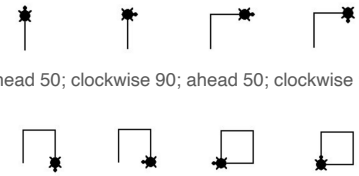






Vector graphics made easy

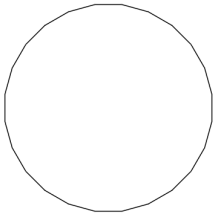
Goal: A readable and transparent interface
(also fun)

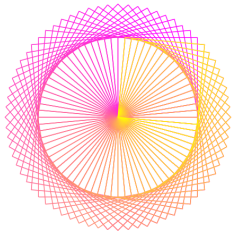


ahead 50; clockwise 90; ahead 50; clockwise 90;

ahead 50; clockwise 90; ahead 50; clockwise 90

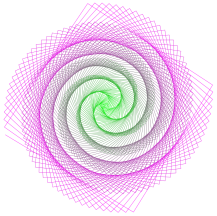
Examples

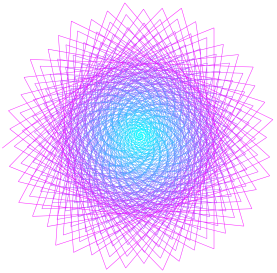




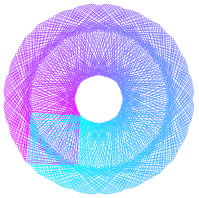
Design Principles

- Moving turtle and pen
- Simple linear commands
  - Loops
  - Variables
  - Abbreviation





Behind the scenes

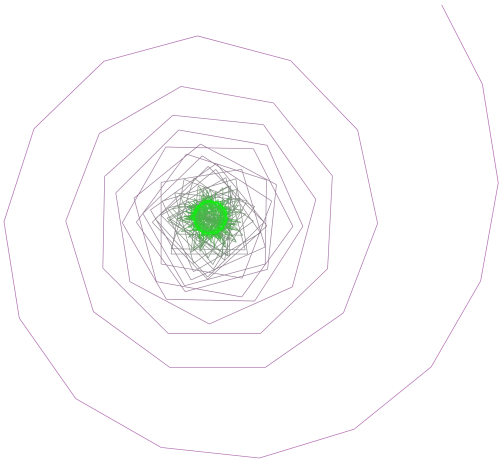


State

- Canvas — line list
- Turtle — movable object
- Pen — attached to Turtle
- Context — variables
- Boundaries — environment

Future work

- Functions
- Lexical scoping
- Interactivity
- Eraser & screen color



F# Natural

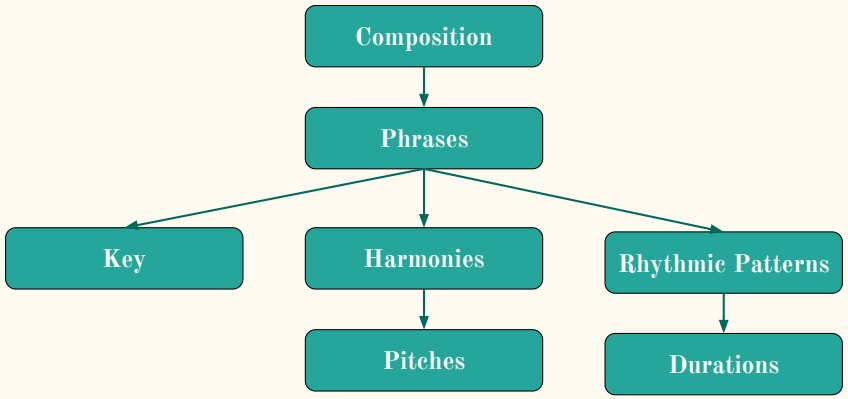
F Natural : A language for music generation

By Margaret Allen and Phoebe Huang

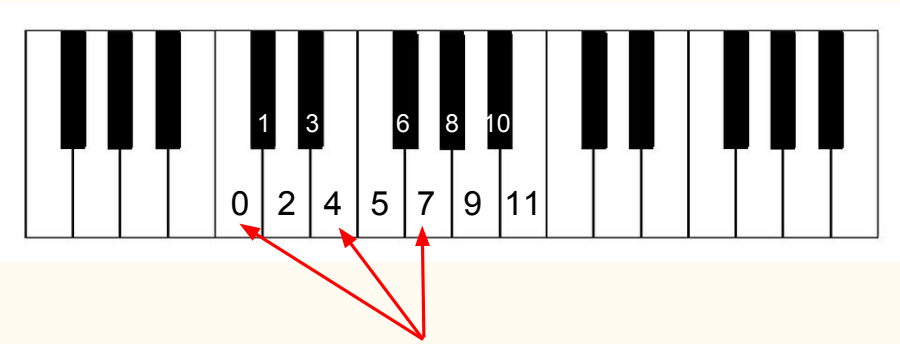
Program Example

```
harmony h1 = {0, 4} -> h2 | END
harmony h2 = {2, 5} -> h1

HarmonySet HS = {h1, h2}

rhythm r1 = (2,2)
rhythm r2 = (4,4,2)
rhythm r3 = (4,8,8,4,4)

RhythmSet RS1 = {r1, r2}
RhythmSet RS2 = {r2, r3}

Phrase p1 = {C, HS, RS1}
Phrase p2 = {G, HS, RS2}

Composition = (p1, p2, p1)
```

Musical Grammar

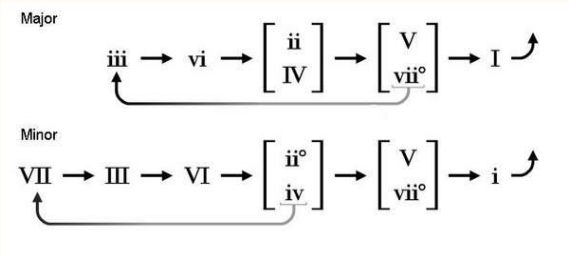


Harmony in Key of C

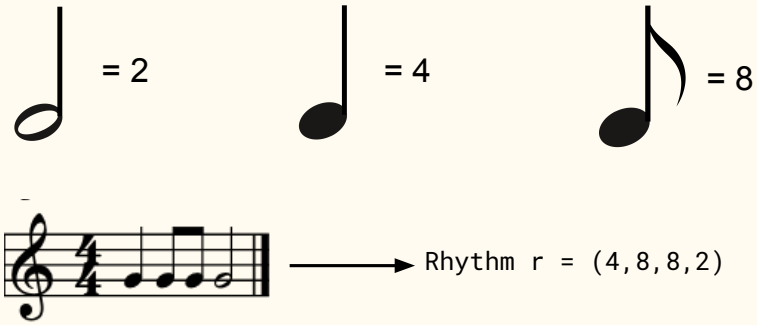


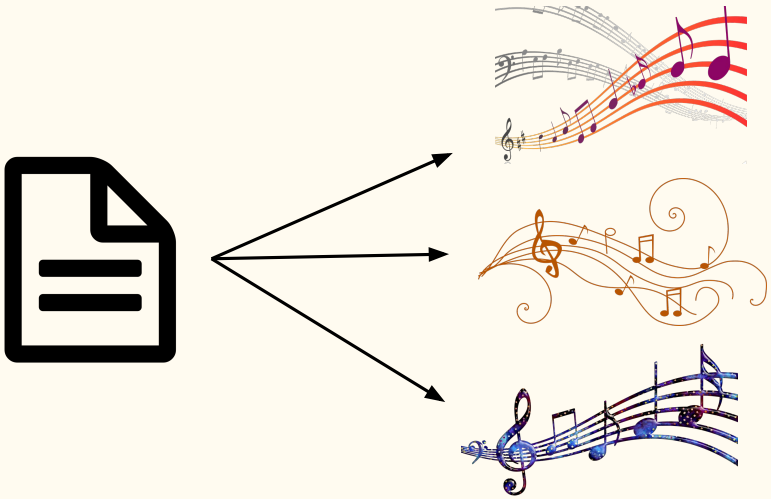
harmony I = {0, 4, 7}

Harmonic Rules



Rhythm





DIGIWEAVE

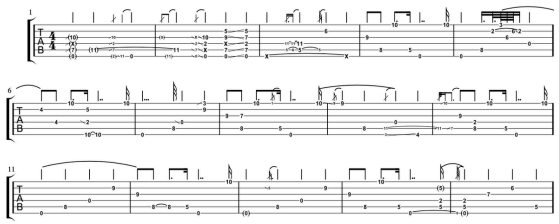
FRIENDSHIP BRACELETS 2.0

Catherine Yeh and Angela Wang

https://catherinesyeh.github.io/digidemo/

Blueberry

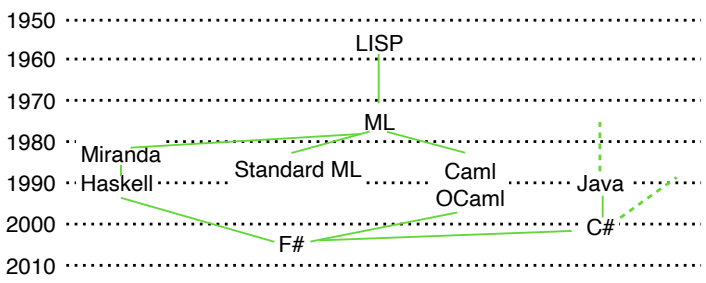
Hugo Hua



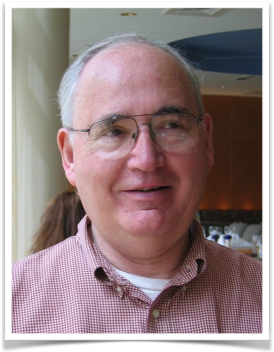
Hugo Hua

https://blueberry.hugohua.com/

ML



ML



- Dana Scott
- Logic of Computable Functions (LCF)
  - Automated proofs!
  - Theorem proving is essentially a "search problem".
  - Proof search is "hard."
    Many problems are NP-Complete.
  - But works "in practice" with the right "tactics"

ML

- Robin Milner
- How to program tactics?
- A "meta-language" is needed
- ML is born (1973)

F#

- Don Syme
- ML is "more fun" than Java or C#.
- Can we use ML instead?
- F# is born (2010).

Logical operators

Logical operators

operation	syntax
and	`&&`
not	`not`
equals	`=`
not equals	`<>`
inequalities	`<, >, <=, >=`

`unit`

`unit` datatype

```
public static void main(String[] args) { … }
```

```
let main args = …
```

`unit` datatype

```
public static void main(String[] args) { … }
```

```
let main(args: string[]) = …
```

Remember: every expression must **return a value**.

A function **can't** return nothing.

`unit` datatype

```
public static void main(String[] args) { … }
```

```
let main(args: string[]) : unit = …
```

Therefore, "nothing" is a thing… called `unit`.

`unit` datatype

```
$ fsharpi

Microsoft (R) F# Interactive version 10.2.3 for F# 4.5
Copyright (c) Microsoft Corporation. All Rights Reserved.

For help type #help;;

> unit;;

  unit;;
  ^^^^

stdin(1,1): error FS0039: The value or constructor 'unit' is
not defined.

> ();;
val it : unit = ()

>
```

How does one obtain a value of `unit`? `()`

You can also `ignore`…

```
> let foo() = 2;;
val foo : unit -> int

> foo();;
val it : int = 2

> ignore (foo());;
val it : unit = ()

> foo() |> ignore;;
val it : unit = ()

>
```

"forward pipe" operator

```
<expr> |> <expr>

foo()  |> ignore
```

By the way…

```
let main(args: string[]) : unit = …
```

By the way…

```
let main(args: string[]) : int = …
```

Lists

Linked List

A **linked list** is a recursive data structure.

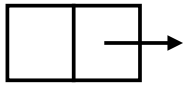
A list is either:

- the **empty list**, or
- a **node**, containing an element and a reference to a list.

Linked List

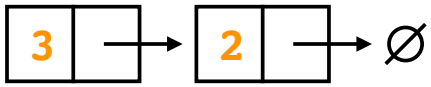
The empty list is defined as **nil** (or `[]`)

Linked List



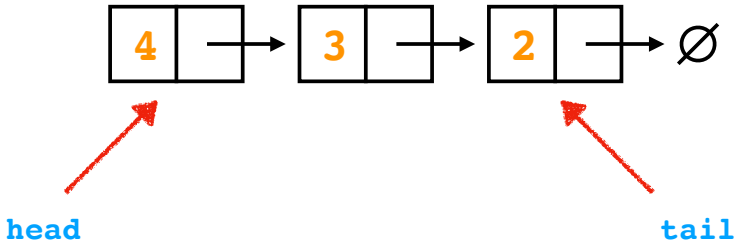
Every other list has at least one list node.

Linked List



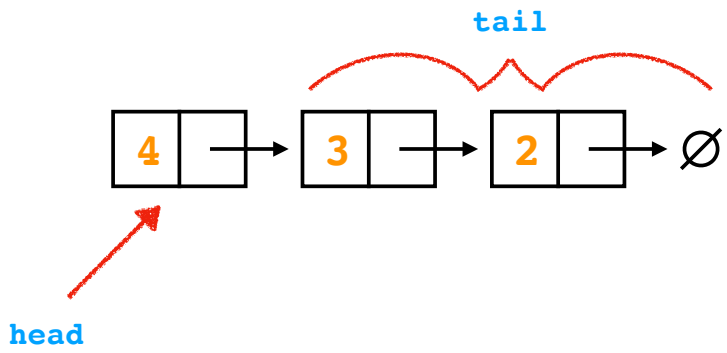
The last node in the list always points to **nil**.

Linked List



A list has parts.

Linked List



A list has parts.

Lists

- Examples
  - `[]` is the empty list
  - `[1; 2; 3; 4]`, `["wombat"; "dingbat"]`
  - ulike LISP, all elements of list **must be same type**
- Operations
  - length `List.length [1;2;3]` ⇒ `3`
  - cons `1::[2;3]` ⇒ `[1; 2; 3]`
  - head `List.head [1;2;3]` ⇒ `1`
  - tail `List.tail [1;2;3]` ⇒ `[2;3]`
  - append `[1;2]@[3;4]` ⇒ `[1; 2; 3; 4]`
  - map `List.map succ [1;2;3]` ⇒ `[2;3;4]`

List types

- `1::2::[] : int list`
  `"wombat"::"numbat"::[] : string list`
- What **type** of list is `[]`?

```
- [];
val it : 'a list
```

- Polymorphic type
  - `'a` is a type variable that represents **any type**
  - `1::[] : int list`
  - `"a"::[] : string list`

Recursive functions

- Note that **recursive** functions must use **`rec`** keyword.
- Not valid:

```
let fact n =
  if n <= 0 then
    1
  else
    n * fact (n - 1)
```

- Instead:

```
let rec fact n =
  if n <= 0 then
    1
  else
    n * fact (n - 1)
```

Functions on Lists

Let's define product…

```
> let rec product nums =
    if (nums = []) then
      1
    else
      (List.head nums)
       * product (List.tail nums);;

val product : int list -> int

> product [5; 2; 3];;
val it : int = 30
```

Pattern matching

```
let rec product nums =
  if (nums = []) then
    1
  else
    (List.head nums)
    * product (List.tail nums)
```

Using **patterns**…

```
let rec product nums =
  match nums with
  | []     -> 1
  | x::xs -> x * product xs
```

Recap & Next Class

Today:

Project ideas

History of ML

F#

Next class:

More F#