CSCI 334:
Principles of Programming Languages

Lecture 14: Project Ideas / Evaluation

Instructor: Dan Barowy

Williams

Announcements

Announcements

No class Thursday

Example: brace language

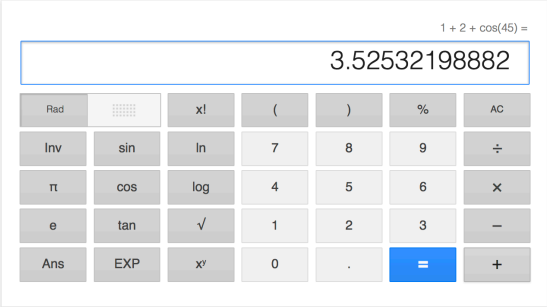
- An ***expression*** is a sequence of ***terms***, consisting of ***at least one term***.
- A ***term*** is either '`aaa`', '`bbb`', or a ***brace expression***.
- A ***brace expression*** is '`{`', followed by an ***expression***, followed by '`}`'.

Lexical vs Dynamic Scope

Inspiration for Projects

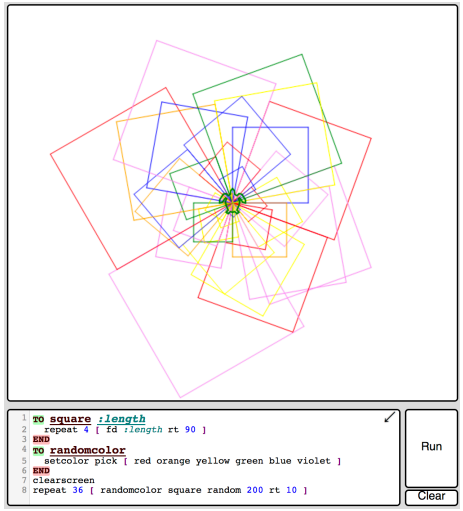
Scientific Calculator

(using infix expressions)



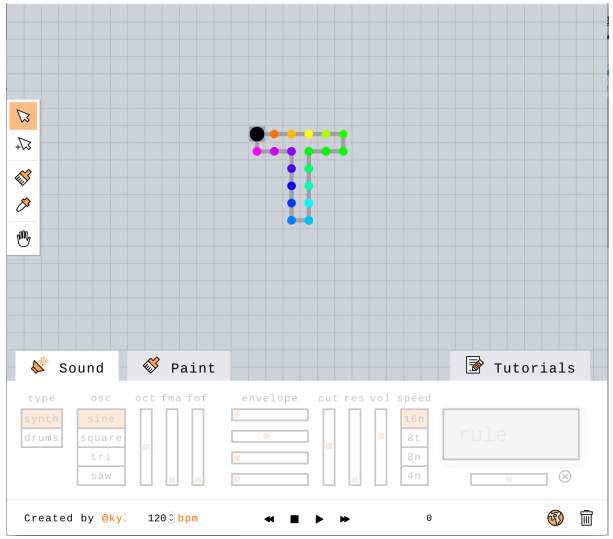
https://www.google.com/search?q=google+calculator

Logo Interpreter



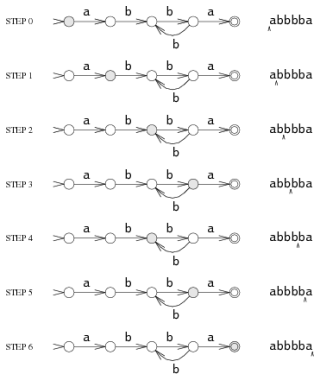
https://www.calormen.com/jslogo/

Turtle Audio



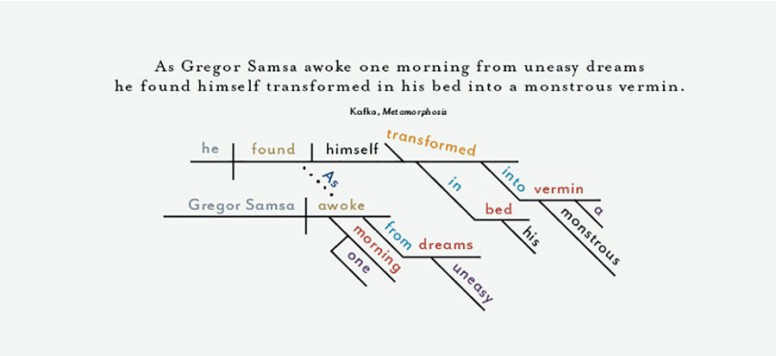
https://turtle.audio

Regular Expressions



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

Auto Sentence Diagramming

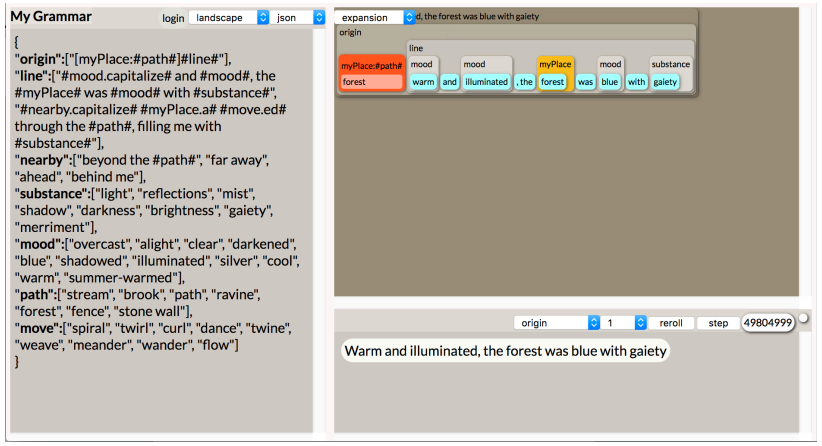


https://www.npr.org/sections/ed/2014/08/22/341898975/a-picture-of-language-the-fading-art-of-diagramming-sentences

BASIC

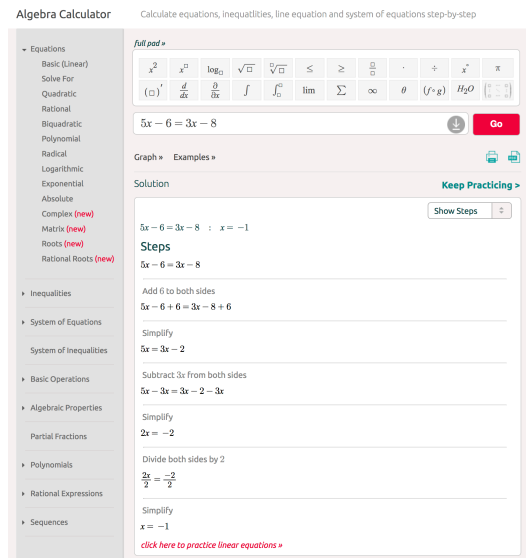
```
710 IF PR(IPS,3)=1 THEN PRINT "Your victim is badly wounded" :: RETURN
720 PRINT "You killed your victim"
730 PR(IPS,1)=0 :: IVW=8+IPS
740 VP(IVW)=IKM :: IPS=0
750 RETURN
760 PRINT "Possible commands:"
770 FOR I=1 TO 15
780 PRINT H$(I)
790 NEXT I
800 RETURN
810 IF IVW<>0 THEN 840
820 PRINT "There is nothing to take    here"
830 RETURN
840 IF BLL=0 AND IKM<25 THEN PRINT "You cannot take things that you do not see"
:: RETURN
850 PRINT "I take the ";V$(IVW);" for you"
860 B(IBMAX)=IVW
870 IBMAX=IBMAX+1
880 VP(IVW)=0
890 IVW=0
900 RETURN
910 INPUT "What would you like to drop? ":D$
920 LNG=LEN(D$)
930 IVWB=0
940 FOR I=1 TO 12
950 IF D$=SEG$(V$(I),1,LNG)THEN IVWB=I :: I=12
960 NEXT I
970 IF IVWB<>0 THEN 1000
980 PRINT "I do not understand you"
```

A BNF parser and generator



http://tracery.io/editor/

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#t-237999

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

Program Evaluation as Reduction

1 + 2 * 3 - 1

(λx.λx.x)(λx.y)

(λx.λx.x)(λx.y)

$\Downarrow \alpha$

$$(\lambda x.\lambda x.x)\,(\lambda x.y)$$

$$\Downarrow \alpha$$

$$(\lambda x.\lambda a.a)\,(\lambda b.y)$$

Goal: alpha-normal form

Goal: alpha-normal form

1. No bound variable uses the same name as any free variable.

Goal: alpha-normal form

1. No bound variable uses the same name as any free variable.
2. No bound variable uses the same name as any other bound variable.

Goal: alpha-normal form

1. No bound variable uses the same name as any free variable.
2. No bound variable uses the same name as any other bound variable.

In other words, all variable names are unique.

Parts

```
e: Expr            An expression
b: Set<char>       Variable bindings
r: Map<char,char>  Renamings
```

```
alphanorm(e: Expr)(b: Set<char>)(r: Map<char,char>)
                : Expr*Set<char>
```

What is passed in; returned

```
alphanorm(e: Expr)(b: Set<char>)(r: Map<char,char>)
                : Expr*Set<char>
```

What is passed in; returned

```
alphanorm(e: Expr)(b: Set<char>)(r: Map<char,char>)
                : Expr*Set<char>
```

Note that we want the set of bindings to persist, therefore it is both *passed in* and *out*.

What is passed in; returned

```
alphanorm(e: Expr)(b: Set<char>)(r: Map<char,char>)
                  : Expr*Set<char>
```

Note that we want the set of bindings to persist, therefore it is both *passed in* and *out*.

But the set of renamings is *scoped*: it is only passed in.

Algorithm

Algorithm

Var(v):

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

App(e1, e2):

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

App(e1, e2):
α-norm e1 & e2 and return App(e1, e2)

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

App(e1, e2):
α-norm e1 & e2 and return App(e1, e2)

Abs(v,e):

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

App(e1, e2):
α-norm e1 & e2 and return App(e1, e2)

Abs(v,e):
if v is already bound, add renaming rule, α-norm e, then return Abs(v', e');

Algorithm

Var(v):
if there is a renaming rule, rename and return renamed Var;
otherwise, return original Var

App(e1, e2):
α-norm e1 & e2 and return App(e1, e2)

Abs(v,e):
if v is already bound, add renaming rule, α-norm e, then return Abs(v', e');
otherwise, return α-norm e and return Abs(v, e')