# Genetic Algorithms and Genetic Programming in Python

Intellovations
Software for Discovery

Eric Floehr
Intellovations, LLC
eric@intellovations.com
(614) 440-0130

# Evolved Virtual Creatures

http://www.youtube.com/watch?v=JBgG_VSP7f8

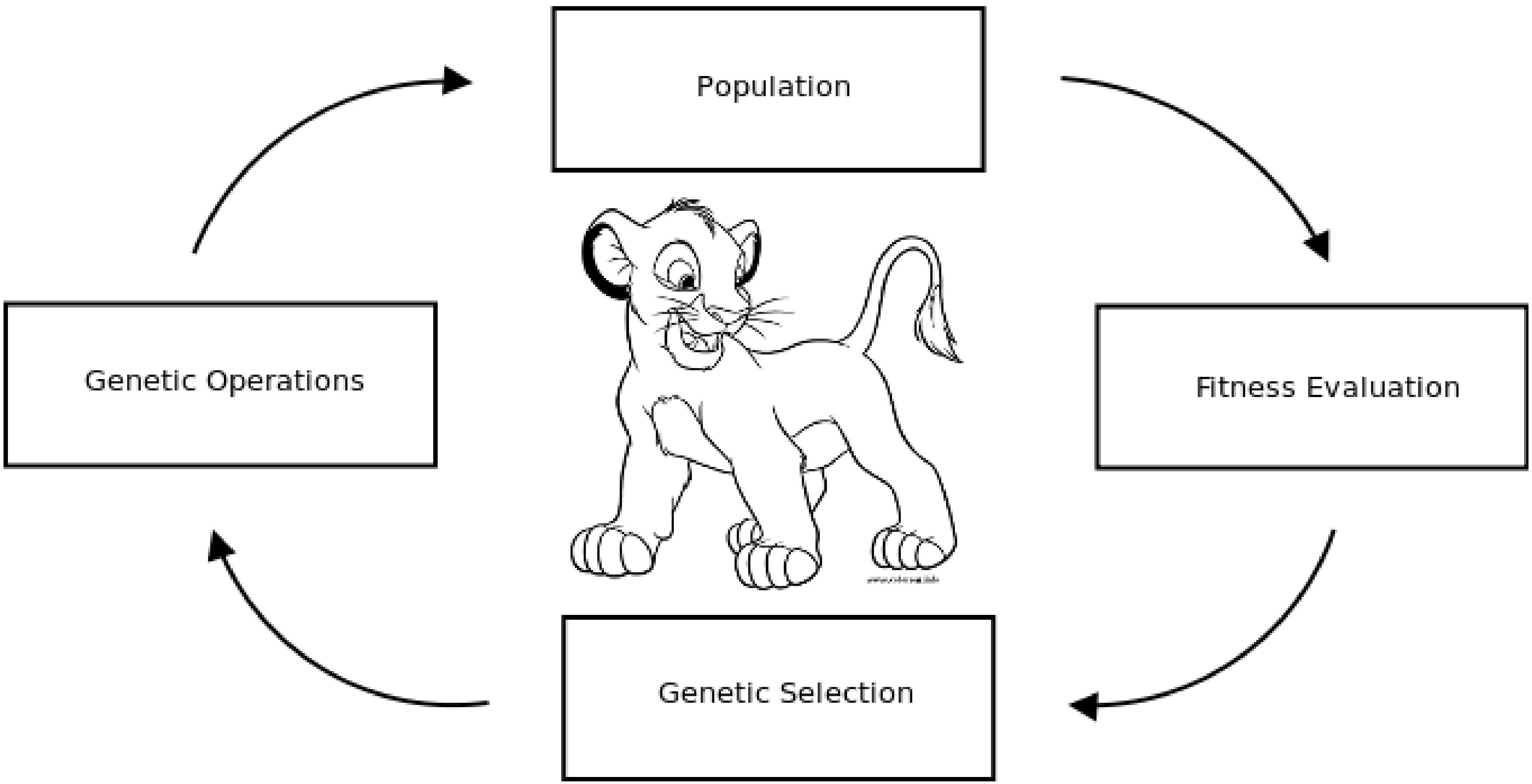
# What are Genetic Algorithms and Genetic Programs?

Search algorithms based on the mechanics of ...

# Natural Selection

# Natural Genetics

# The Mechanics (Circle of Life)



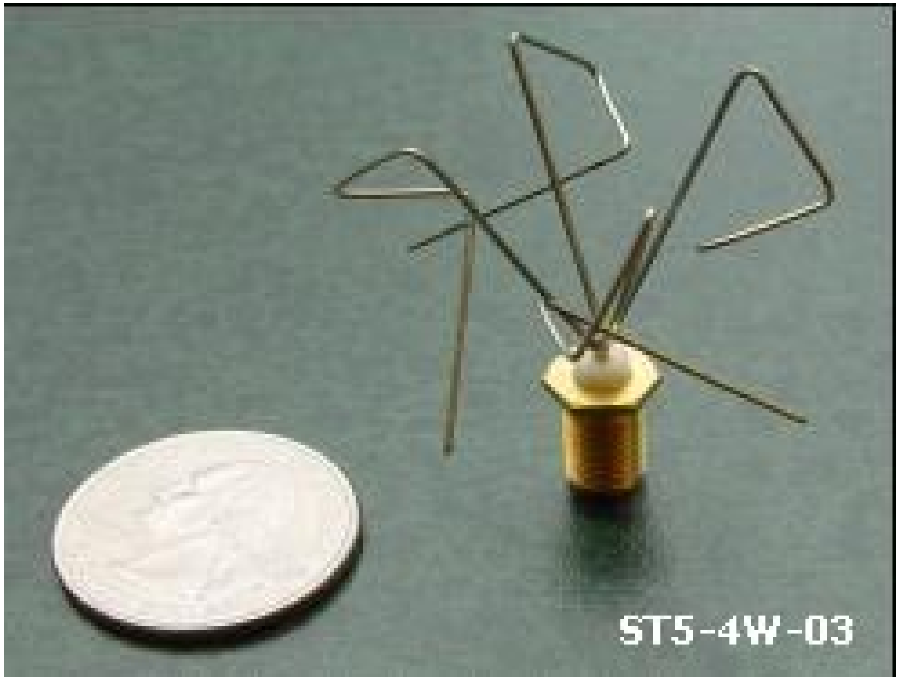
# John Holland, University of Michigan

# John Koza, University of Michigan, Stanford

# They seem to do well when...

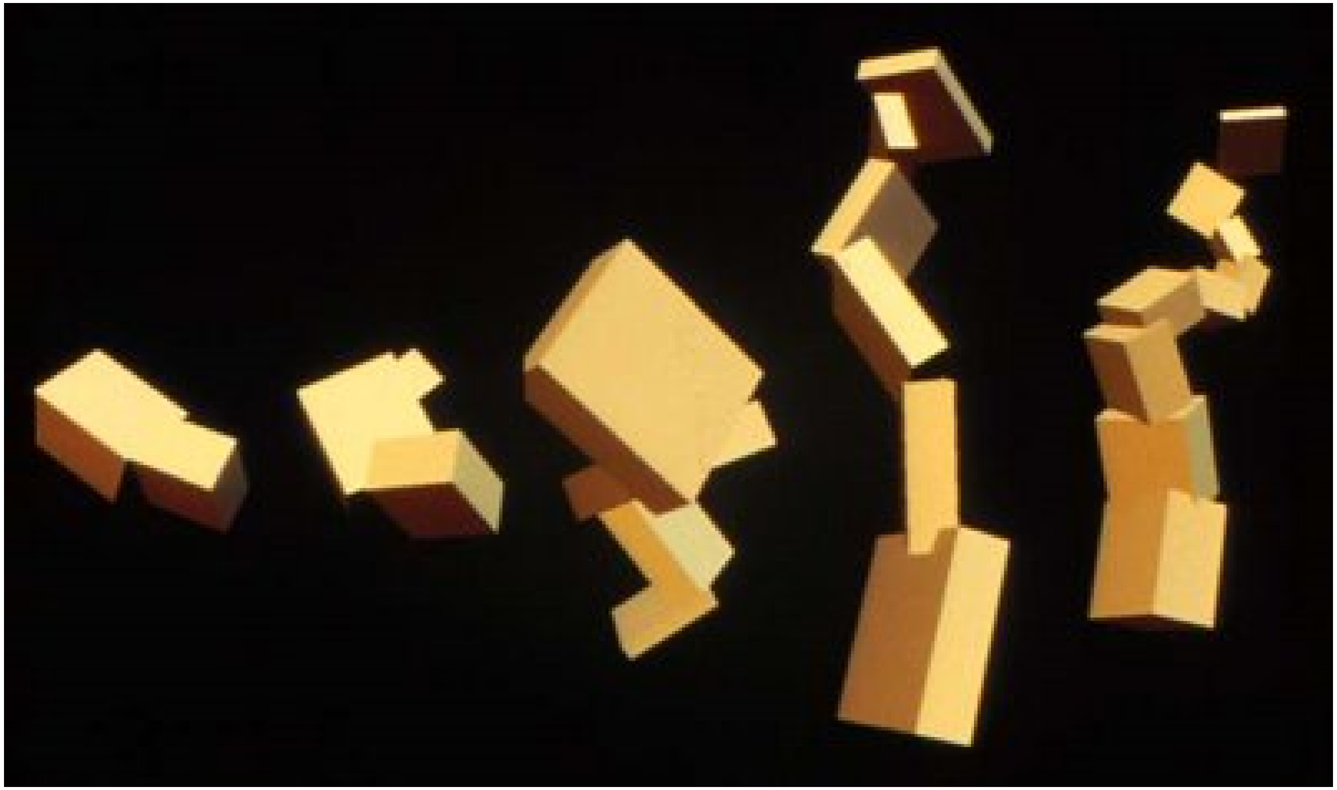
- There is a huge solution space
- The solution space is dynamic, non-linear, or otherwise complex
- The solution space is poorly understood
- Domain knowledge is scarce or difficult to encode
- No mathematical analysis is available
- Fitness, payoff, or suitability of a solution can be determined
- Traditional search methods fail

# Antenna Design



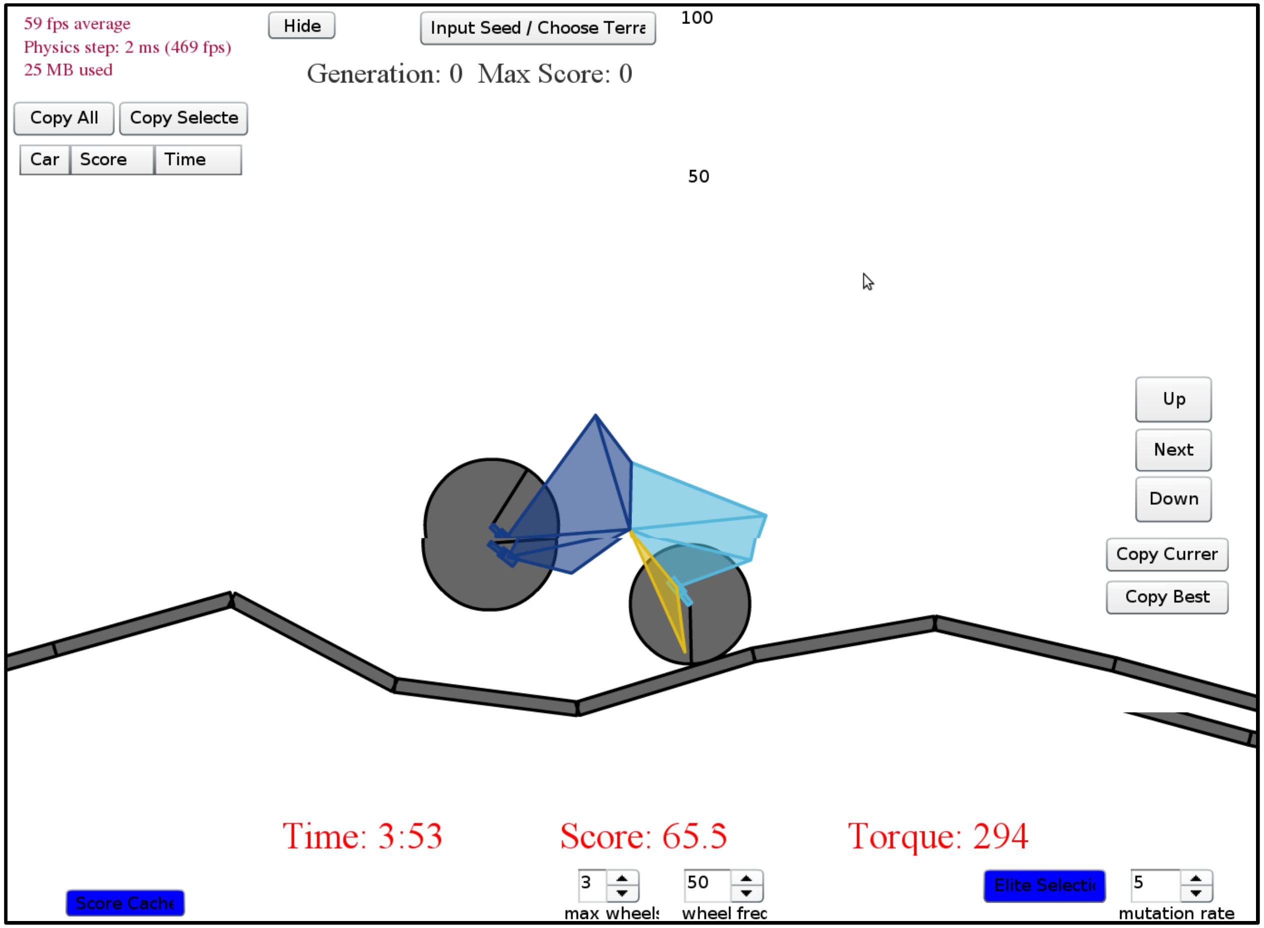

http://ti.arc.nasa.gov/projects/esg/research/antenna.htm

# Locomotion



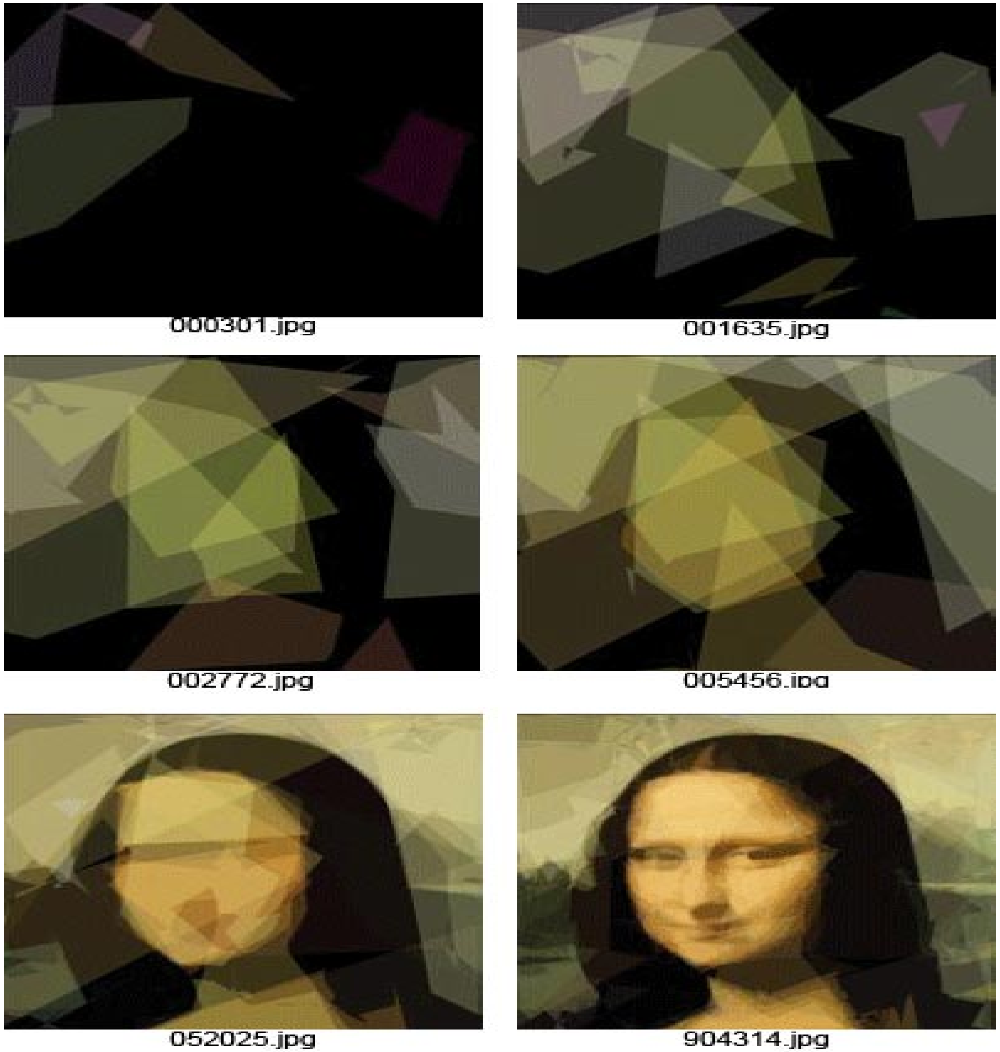
http://www.karlsims.com/evolved-virtual-creatures.html

# Evolving Cars



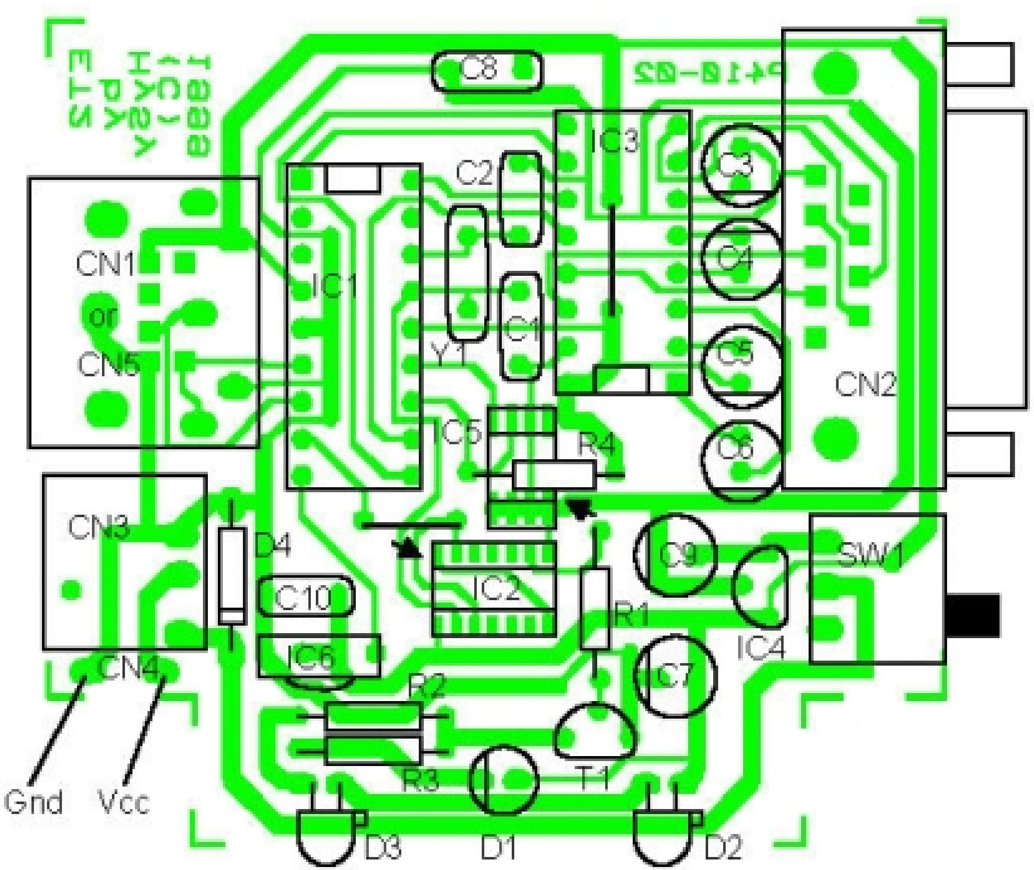

http://boxcar2d.com/

# Art



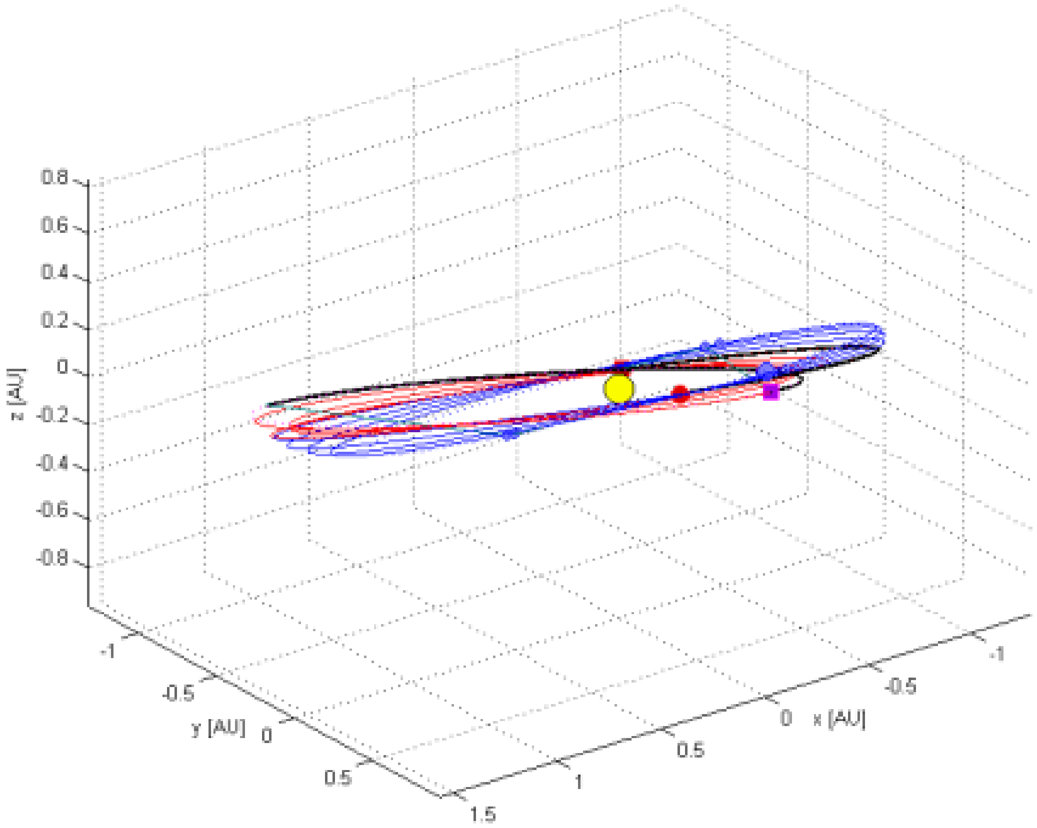

http://rogeralsing.com/2008/12/07/genetic-programming-evolution-of-mona-lisa/

# Minimize connection length




http://dx.doi.org/10.1115/1.2792119

# Spacecraft Tragectory Design




http://www.astos.de/solutions/space/interplanetary

# The “Humie” Awards

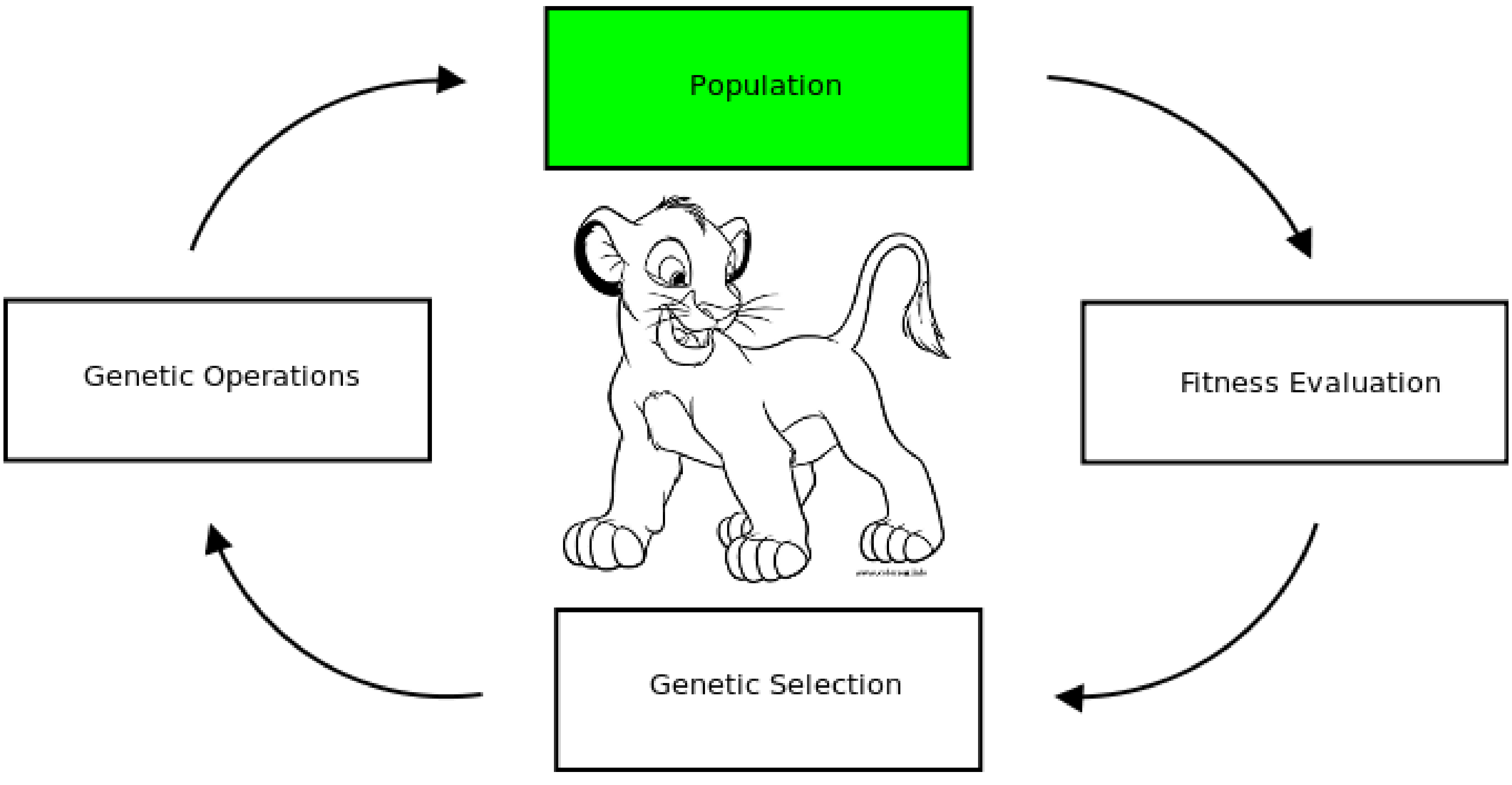
- Evolve patches to C programs
- Evolve efficient search algorithm for mate-in-N problem
- Diagnosing prostate cancer using infrared spectroscoping imaging
- Extracting ellipses from an image using GP
- Automated test program generation for microprocessor test and validation

# PyEvolve

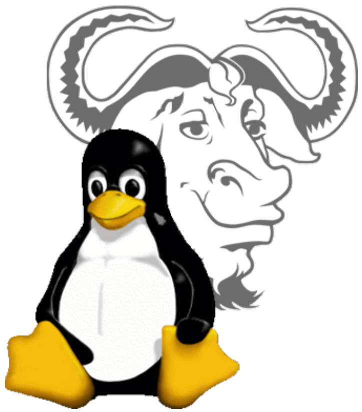
- GA/GP Library for Python
- Uses multiprocessing
- Fast with PyPy!
  - http://pyevolve.sourceforge.net/wordpress/?p=1189
- Currently released version 0.5
  - Don't Use
- Better to use PyEvolve 0.6RC1
  - Should be released near the end of March

What are the “mechanics” of natural selection and natural genetics?

# The Circle of Life



# Genotype, Phenotype, and Genome



“blueprint”


“Instances”

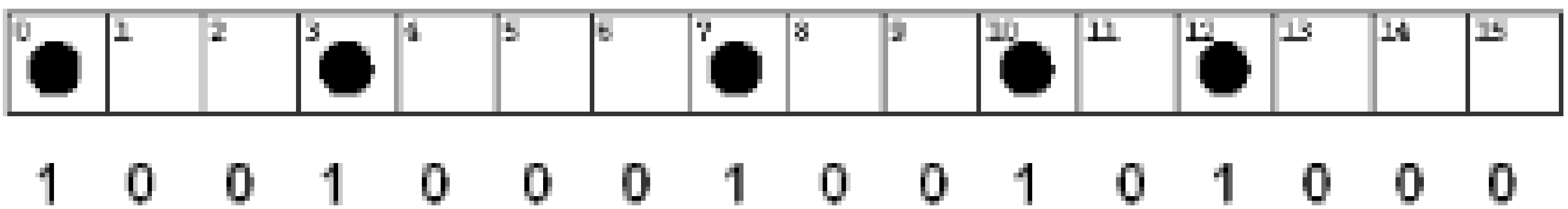

“Physical manifestation”

# Pyevolve Built-In Genotypes

- One-Dimensional Binary String
- Two-Dimensional Binary String
- One-Dimensional List
- Two-Dimensional List
- Tree
- Genetic Program (Tree specific to GP)
- You can roll your own!

# 1-D Binary String

```
from pyevolve import G1DBinaryString

genotype = G1DBinaryString.G1DBinaryString(16)
```



# 2-D Binary String

```
from pyevolve import G2DBinaryString

genotype = G1DBinaryString.G2DBinaryString(12, 36)
```

```
        4    5    6    7    8    9   10   11   12   13   14   15   16   17   18   19   20   21
  1:    H    H    H    H    H    H    D    H    H    S    D    H    D    H    S    S    S    S
  2:    H    S    H    D    H    H    H    H    D    D    H    S    S    S    S    S    S    S
  3:    D    S    S    H    H    H    D    H    H    H    H    H    D    S    S    S    S    S
  4:    D    H    H    D    H    H    H    H    H    D    S    H    S    H    S    S    S    S
  5:    D    H    H    S    H    H    H    D    D    H    H    S    H    S    S    S    S    S
  6:    H    H    H    H    H    H    D    H    H    D    H    D    S    S    S    S    S    S
  7:    S    H    H    D    H    H    D    H    H    H    S    S    S    S    S    S    S    S
  8:    H    D    H    D    D    D    H    D    H    D    D    D    H    S    S    S    S    S
  9:    S    S    D    D    H    H    H    D    H    H    D    H    H    S    S    S    S    S
 10:    H    H    D    D    H    D    H    H    H    H    H    S    H    S    S    S    S    S
 11:    S    H    H    H    H    H    H    D    D    S    H    H    S    S    S    S    S    S
 12:    D    H    H    H    H    H    H    H    D    H    D    H    S    S    S    S    S    S

  B:    H    H    H    D    H    H    H    H    H    H    H    H    S    S    S    S    S    S

       5H   8H   9H   6D  11H  10H   8H   8H   8H   6H   4H   6H   6S  10S  12S  12S  12S  12S
       4D   3S   2D   5H   1D   2D   4D   4D   4D   4D   3D   4S   4H   2H
       3S   1D   1S   1S                           2S   2S   2D   2D
```

# 1-D and 2-D Lists

```
from pyevolve import G1DList, G2DList

genotype = G1DList.G1DList(140)
genotype = G2DList.G2DList(8, 5)
```

“Alleles” are the object types the list is made of... by default integers
You can pass in a function to construct the list with whatever object you want

```
genotype.setParams(rangemin=-5.12, rangemax=5.13)
genotype.initializator.set(Initializators.G1DListInitializatorReal)
```

# Tree and Genetic Program

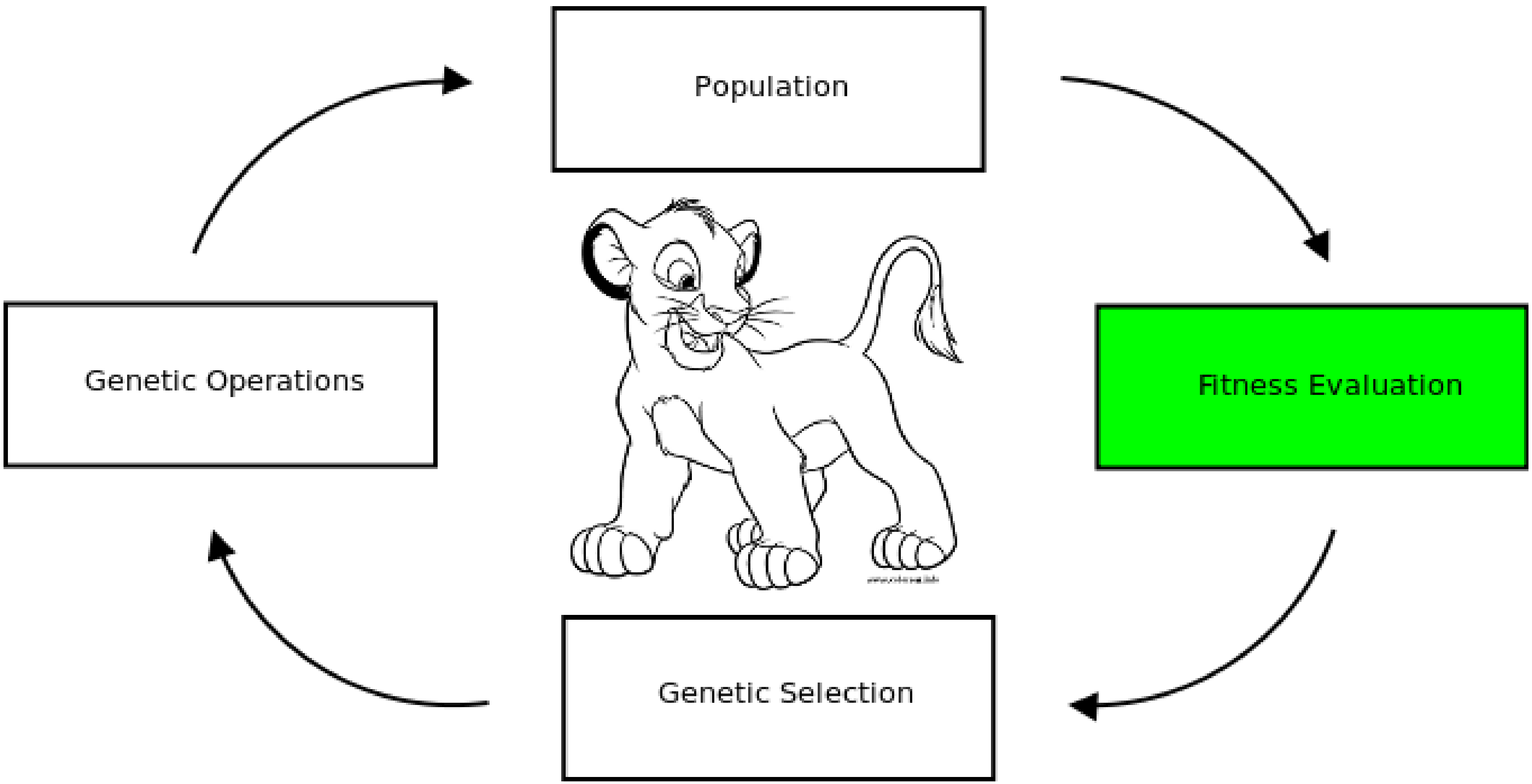
```
from pyevolve import GTree

genotype = GTree.GTree()

gp_genotype = GTree.GTreeGP()
```

# Tree and Genetic Program

```
from pyevolve import GTree

genotype = GTree.GTree()

gp_genotype = GTree.GTreeGP()
```

Now that we have our genotype, how is it expressed, and how do we know how “good” instances of the genotype are?

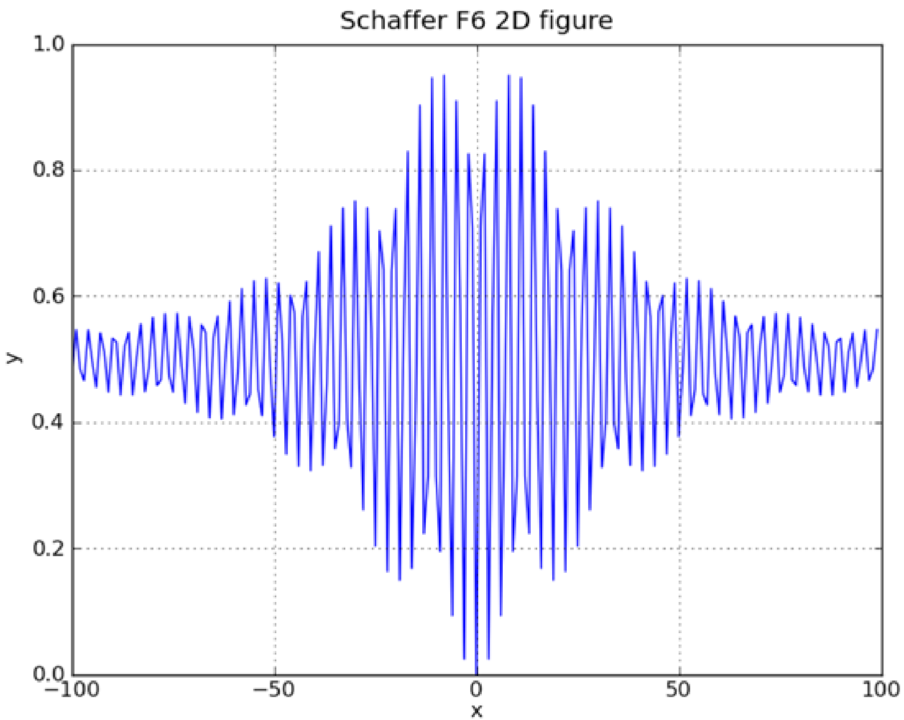
# The Circle of Life



# Fitness

# Find the global minima of Schaffer F6

$$f(x,y) = 0.5 + \frac{\left(\sin\sqrt{(x^2+y^2)}\right)^2 - 0.5}{[1.0 + 0.001(x^2+y^2)]^2}$$



The genotype is a 2D point, i.e. (x,y)

Phenotype is how the (x,y) point manifests itself in its environment, in this case f(x,y)

Genome (or chromosomes) is a specific instance of an (x,y) point, i.e. (3, -2.5)

Fitness is how well the point does in its environment.  In this case, how close f(x,y) is to the global minimum value (in this case, zero)

$$f(x,y) = 0.5 + \frac{\left(\sin\sqrt{(x^2+y^2)}\right)^2 - 0.5}{[1.0 + 0.001(x^2+y^2)]^2}$$

# The Fitness Function

```
def schafferF6(genome):
   x2y2 = genome[0]**2 + genome[1]**2
   t1 = math.sin(math.sqrt(x2y2));
   t2 = 1.0 + 0.001*(x2y2);
   score = 0.5 + (t1**2 - 0.5)/(t2*t2)
   return score
```

```
genotype = G1DList.G1DList(2)
genotype.setParams(rangemin=-100.0, rangemax=100.0,
                bestrawscore=0.0000, rounddecimal=4)
genotype.initializator.set(Initializators.G1DListInitializatorReal)
```

```
genotype.evaluator.set(schafferF6)
```

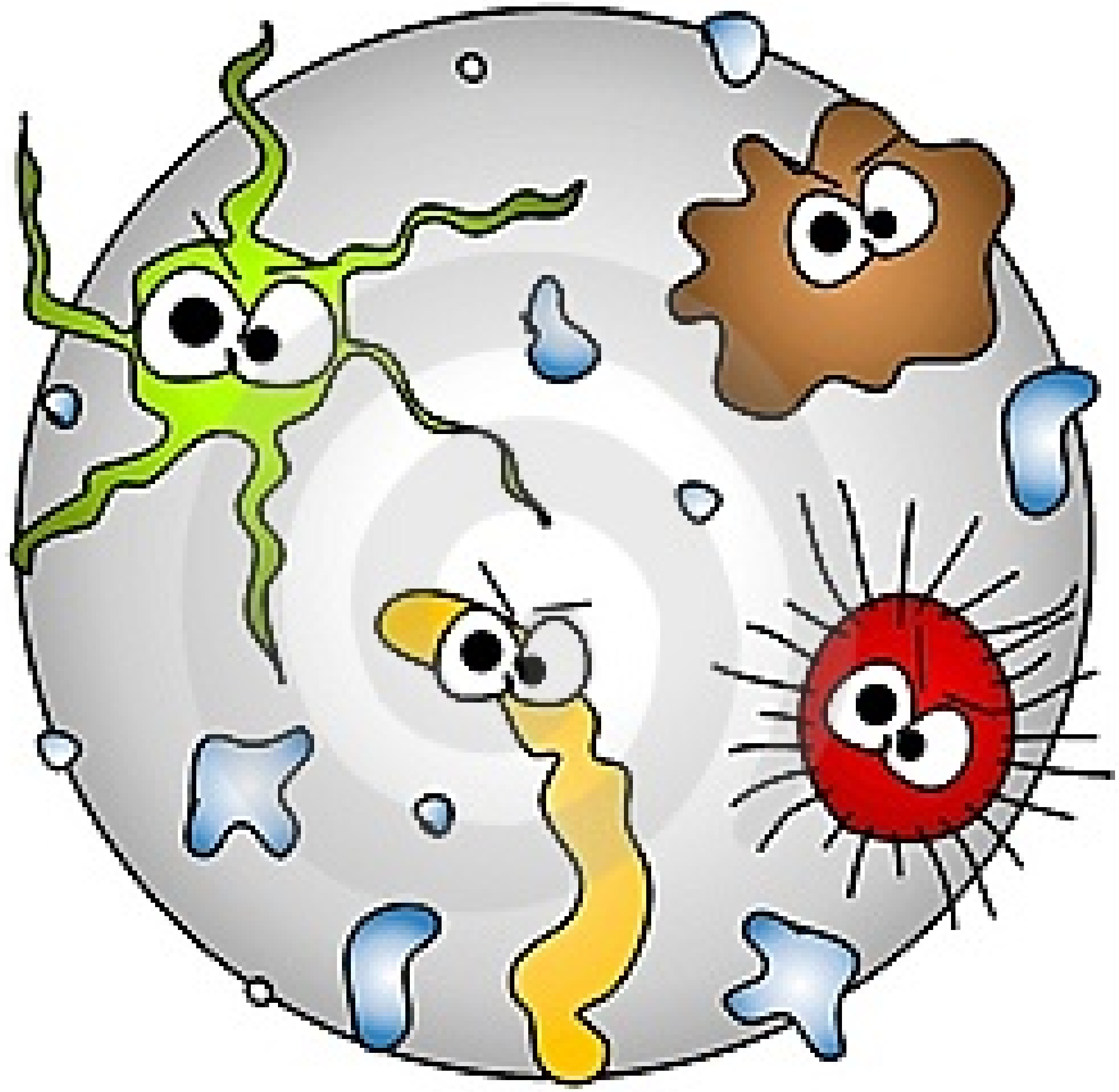
# The Fitness Function

```
def schafferF6(genome):
   x2y2 = genome[0]**2 + genome[1]**2
   t1 = math.sin(math.sqrt(x2y2));
   t2 = 1.0 + 0.001*(x2y2);
   score = 0.5 + (t1**2 - 0.5)/(t2*t2)
   return score
```

```
genotype = G1DList.G1DList(2)
genotype.setParams(rangemin=-100.0, rangemax=100.0,
                bestrawscore=0.0000, rounddecimal=4)
genotype.initializator.set(Initializators.G1DListInitializatorReal)
```

```
genotype.evaluator.set(schafferF6)
```

# The Petri Dish



# The GA Engine

```
from pyevolve import GSimpleGA

… create genome …
```

```
ga = GSimpleGA.GSimpleGA(genome, seed=123)
ga.setMinimax(Consts.minimaxType["minimize"])
ga.evolve(freq_stats=1000)
print ga.bestIndividual()
```

Output:

```
Gen. 0 (0.00%): Max/Min/Avg Fitness(Raw) [0.60(0.82)/0.37(0.09)/0.50(0.50)]
Gen. 1000 (12.50%): Max/Min/Avg Fitness(Raw) [0.30(0.97)/0.23(0.01)/0.25(0.25)]
Gen. 2000 (25.00%): Max/Min/Avg Fitness(Raw) [0.21(0.99)/0.17(0.01)/0.18(0.18)]
Gen. 3000 (37.50%): Max/Min/Avg Fitness(Raw) [0.26(0.99)/0.21(0.00)/0.22(0.22)]

     Evolution stopped by Termination Criteria function !

Gen. 3203 (40.04%): Max/Min/Avg Fitness(Raw) [0.30(0.99)/0.23(0.00)/0.25(0.25)]
Total time elapsed: 14.357 seconds.

- GenomeBase
     Score:          0.000005
     Fitness:        0.232880

- G1DList
     List size:      2
     List:           [0.0020881039453384299, 0.00043589670629584631]
```

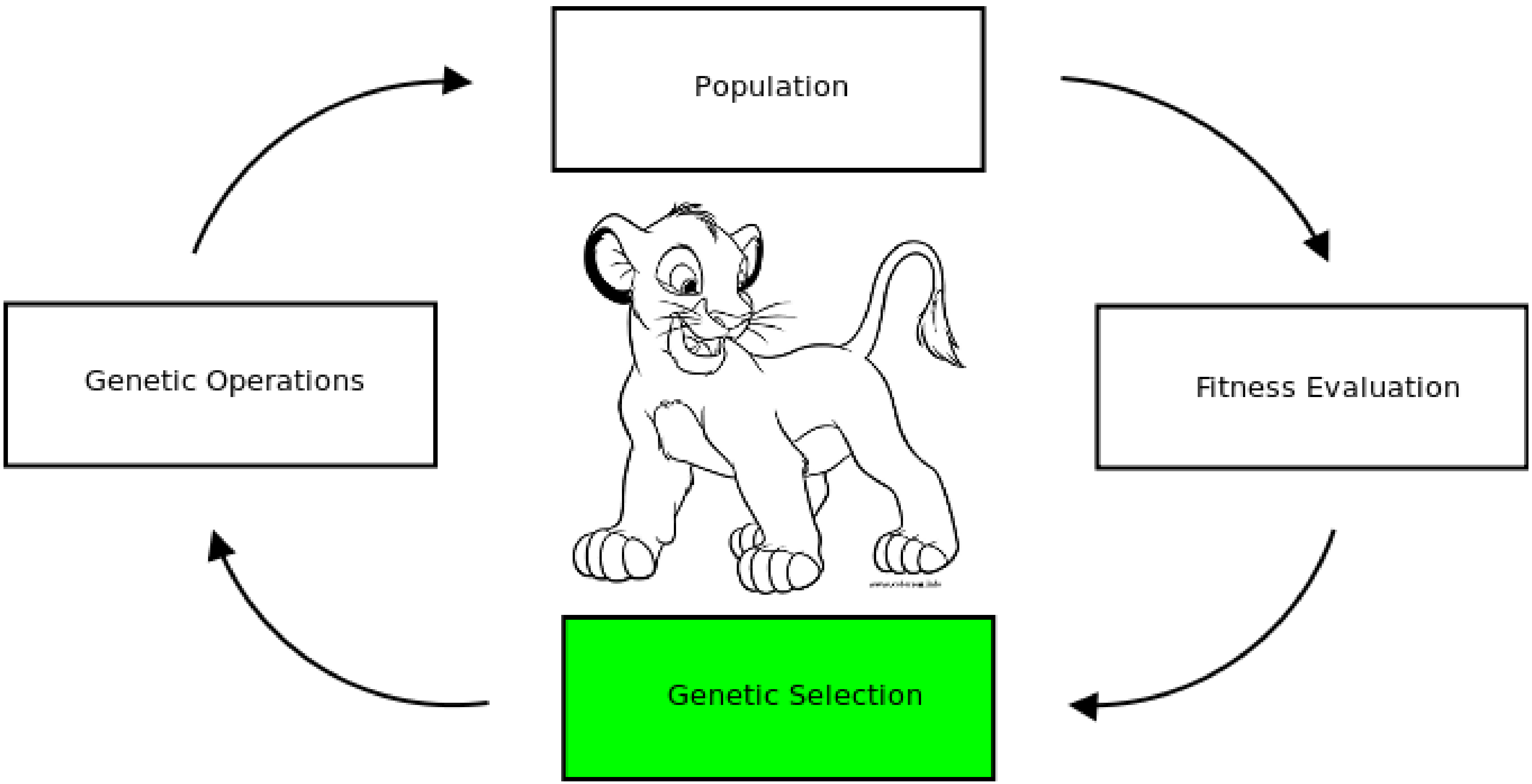
# The GA Engine

```
from pyevolve import GSimpleGA

… create genome …

ga = GSimpleGA.GSimpleGA(genome, seed=123)
ga.setMinimax(Consts.minimaxType["minimize"])
ga.evolve(freq_stats=1000)
print ga.bestIndividual()
```

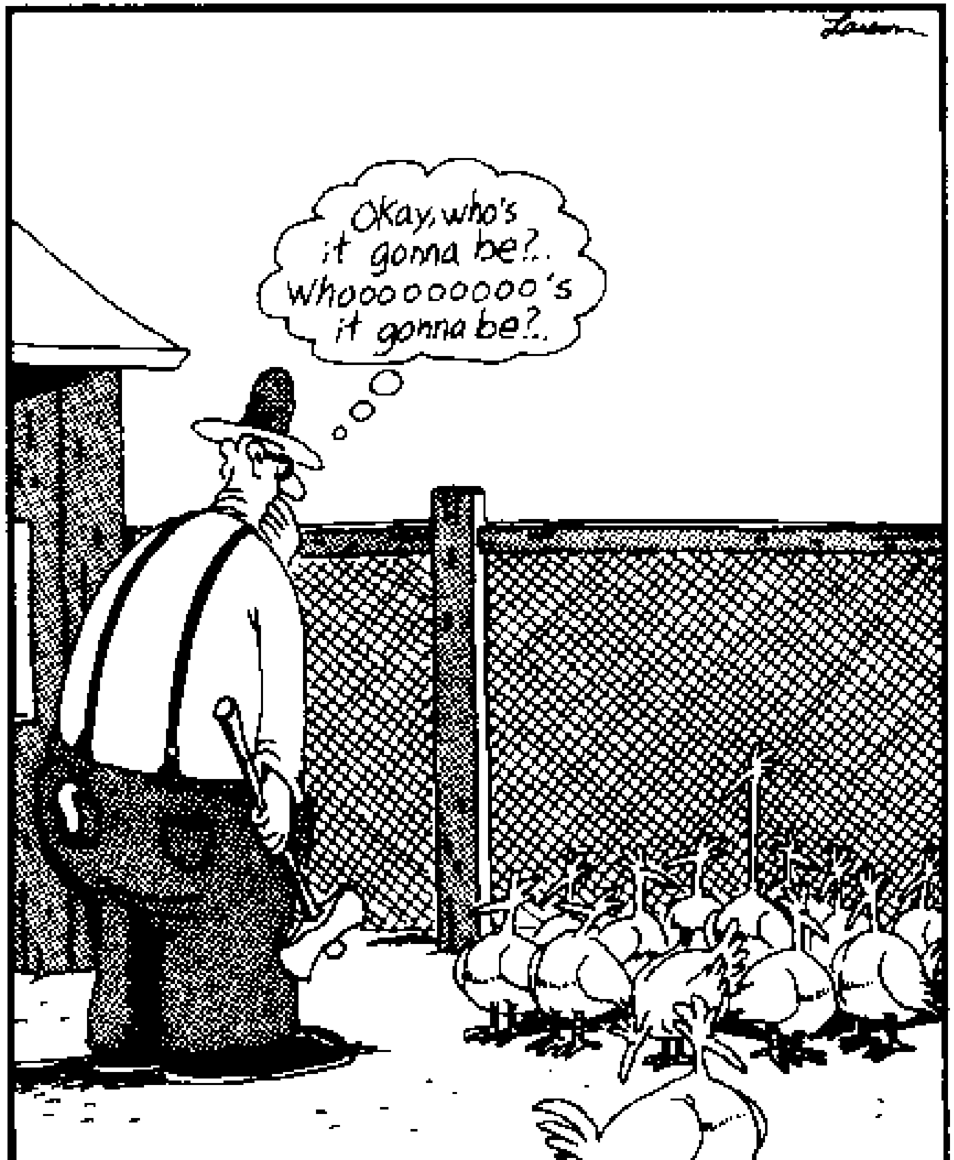
Output:

```
Gen. 0 (0.00%): Max/Min/Avg Fitness(Raw) [0.60(0.82)/0.37(0.09)/0.50(0.50)]
Gen. 1000 (12.50%): Max/Min/Avg Fitness(Raw) [0.30(0.97)/0.23(0.01)/0.25(0.25)]
Gen. 2000 (25.00%): Max/Min/Avg Fitness(Raw) [0.21(0.99)/0.17(0.01)/0.18(0.18)]
Gen. 3000 (37.50%): Max/Min/Avg Fitness(Raw) [0.26(0.99)/0.21(0.00)/0.22(0.22)]

   Evolution stopped by Termination Criteria function !

Gen. 3203 (40.04%): Max/Min/Avg Fitness(Raw) [0.30(0.99)/0.23(0.00)/0.25(0.25)]
Total time elapsed: 14.357 seconds.

- GenomeBase
     Score:          0.000005
     Fitness:        0.232880

- G1DList
     List size:      2
     List:           [0.0020881039453384299, 0.00043589670629584631]
```

# The Circle of Life



# Selection




© Gary Larson, Farside

# Pyevolve Built-In Selection Operators

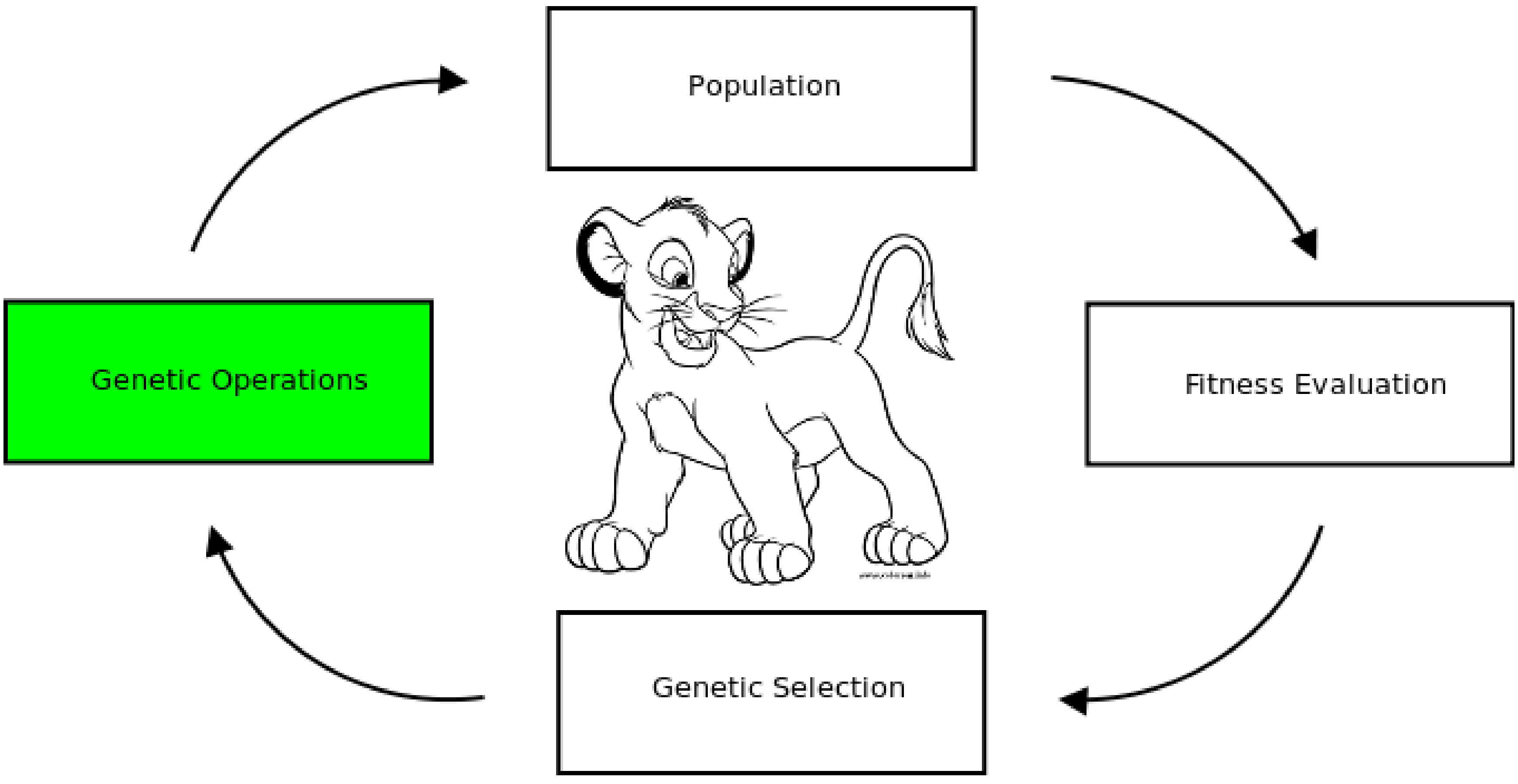
- Rank Selection
  - Choose the best (default)
- Uniform Selection
  - Choose at random
- Tournament Selection
  - Choose best from a random subset of population
- Roulette Wheel Selection
  - More fit more likely to be chosen

# Setting the selector

```
from pyevolve import Selectors

ga = GSimpleGA.GSimpleGA(genome)
ga.selector.set(Selectors.GRouletteWheel)
```

ga.selector is a “FunctionSlot.”  It can accept any number of functions that will be used in order.  ga.evaluator is another.

# The Circle of Life

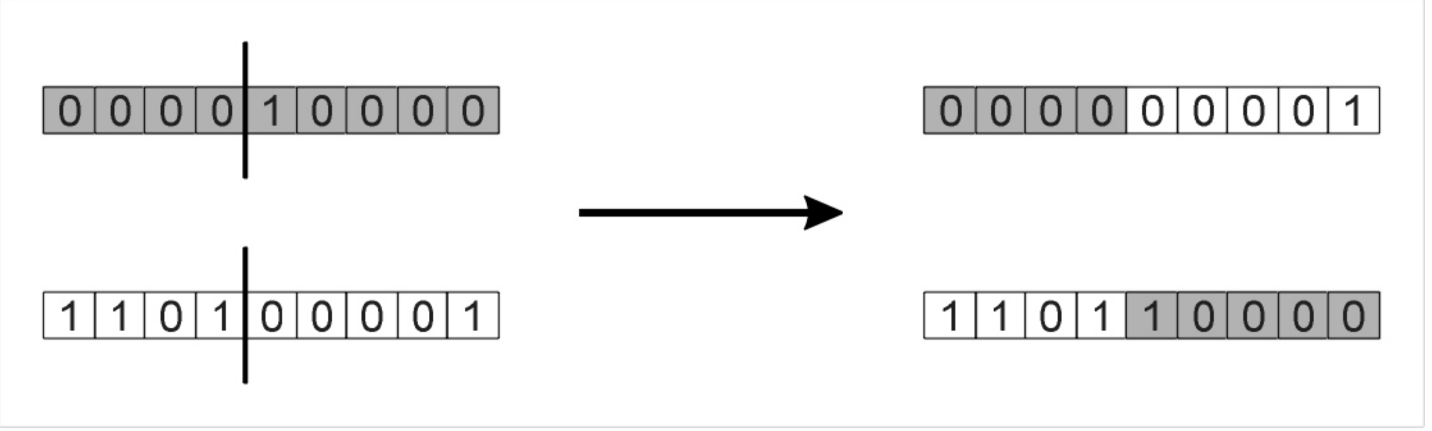


# Crossover

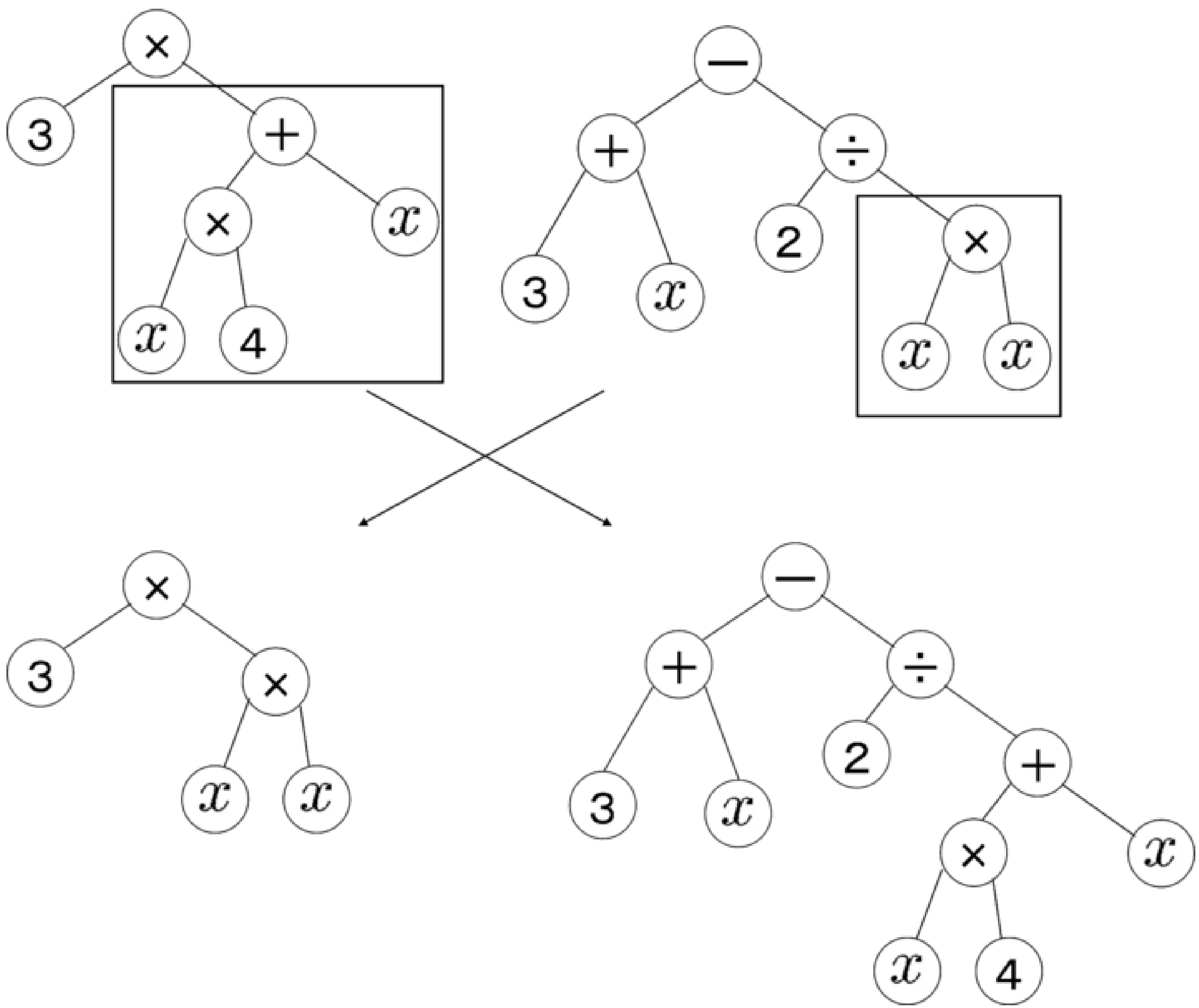

For Ages 4 and up.
STAR TREK
TEENAGE MUTANT NINJA
TURTLES®
BEAM UP AND COLLECT ALL THE
CLASSIC KOWABUNGA CHARACTERS!
CHIEF ENGINEER
MICHAELANGELO™
FIRST OFFICER
DONATELLO™
CAPTAIN
LEONARDO™
CHIEF MEDICAL
OFFICER RAPHAEL™

Take two (or more) individuals and combine them in some way to create children

# Single Point Crossover



# Single Point Crossover

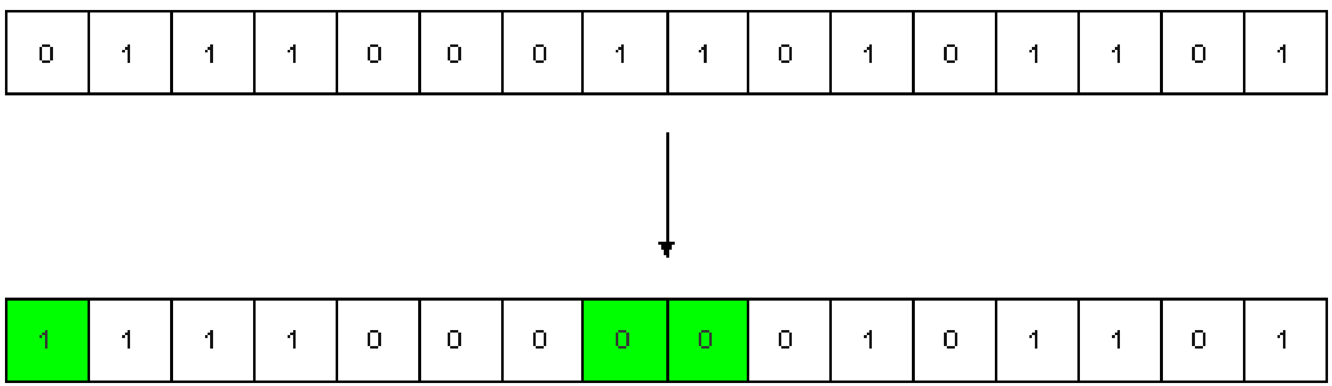


# Pyevolve Built-In Crossover Operators

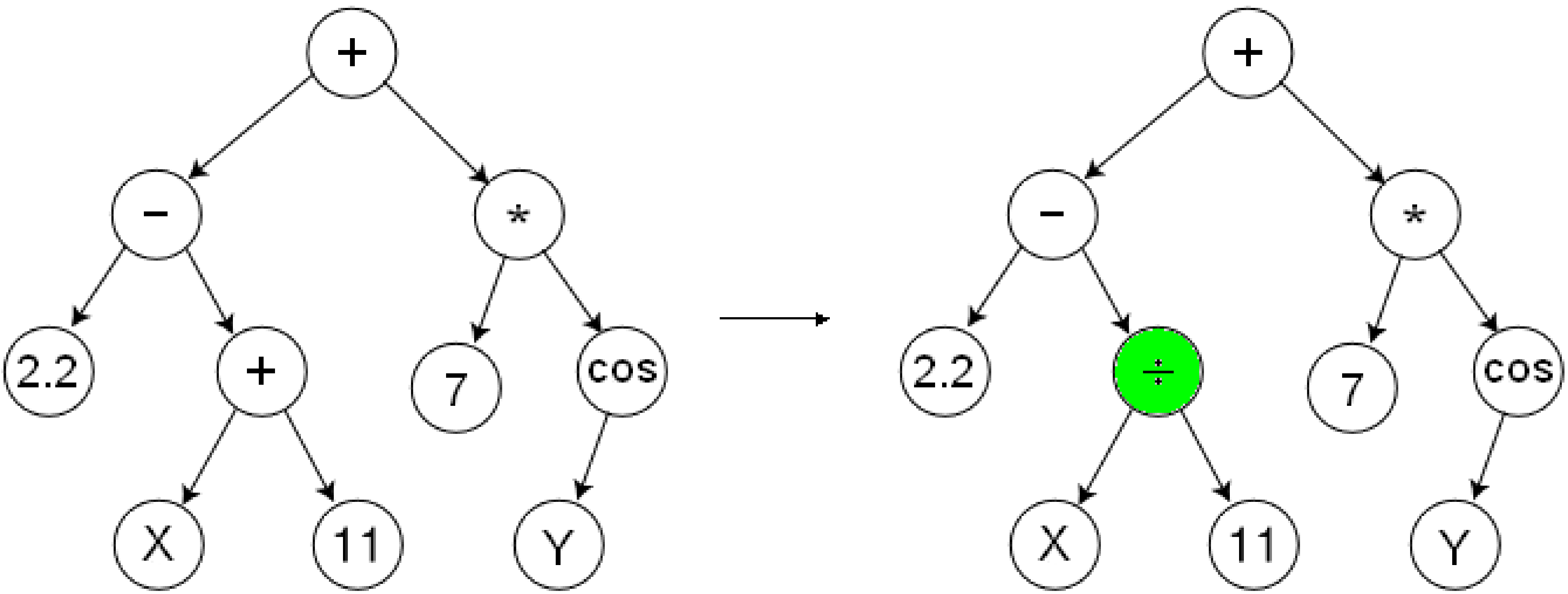
- 1D Binary String
  - Single Point Crossover, Two Point Crossover, Uniform Crossover
- 1D List
  - Single Point Crossover, Two Point Crossover, Uniform Crossover, OX Crossover, Edge Recombination Crossover, Cut and Crossfill Crossover, Real SBX Crossover
- 2D List
  - Uniform Crossover, Single Vertical Point Crossover, Single Horizontal Point Crossover
- 2D Binary String
  - Uniform Crossover, Single Vertical Point Crossover, Single Horizontal Point Crossover
- Tree
  - Single Point Crossover, Strict Single Point Crossover
- GP Tree
  - Single Point Crossover

# Mutation

# Binary Mutation



# Tree Mutation



# Pyevolve Built-In Mutation Operators

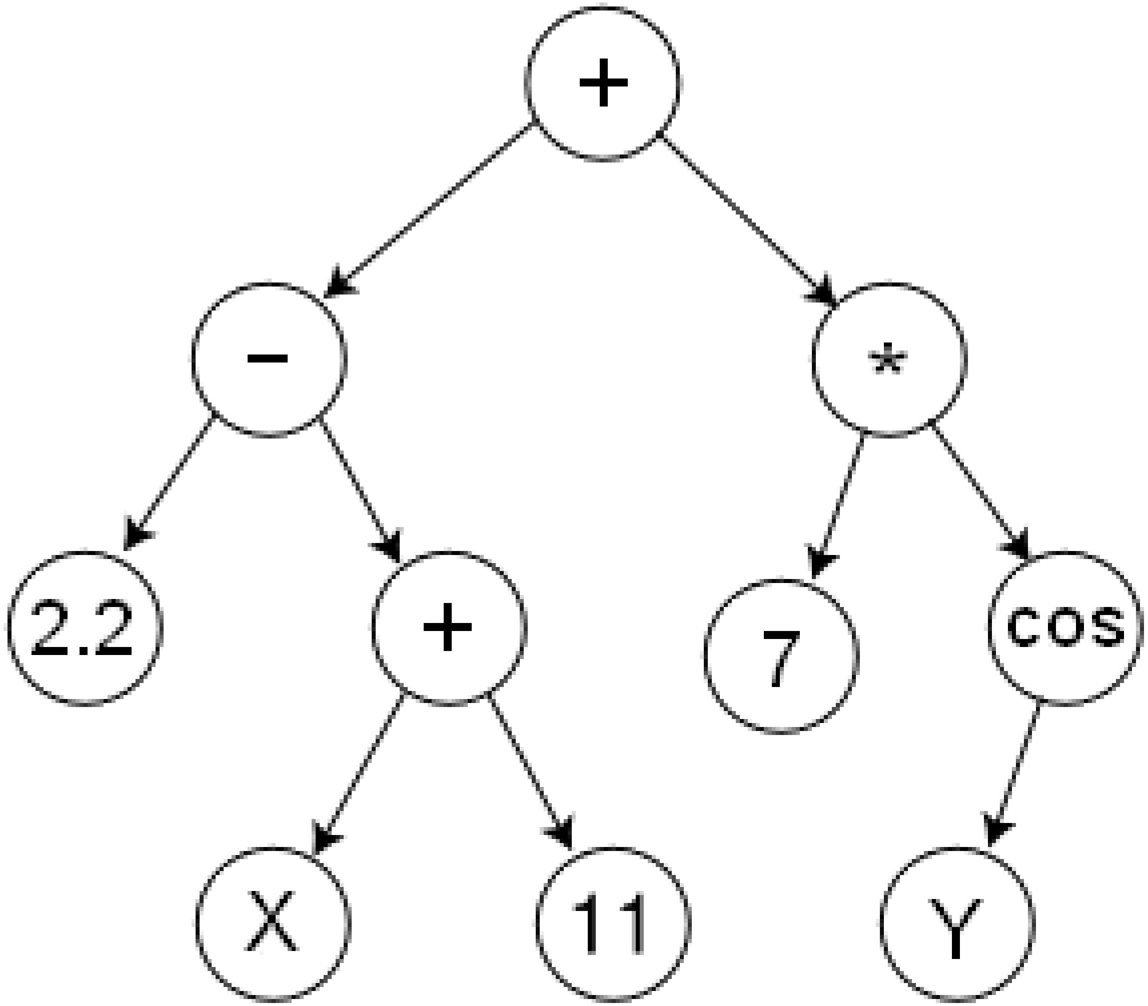
- 1D Binary String
  - Swap Mutator, Flip Mutator
- 2D Binary String
  - Swap Mutator, Flip Mutator
- 1D List
  - Swap Mutator, Integer Range Mutator, Real Range Mutator, Integer Gaussian Mutator, Real Gaussian Mutator, Integer Binary Mutator, Allele Mutator, Simple Inversion Mutator
- 2D List
  - Swap Mutator, Integer Gaussian Mutator, Real Gaussian Mutator, Allele Mutator, Integer Range Mutator
- Tree
  - Swap Mutator, Integer Range Mutator, Real Range Mutator, Integer Gaussian Mutator, Real Gaussian Mutator
- GP Tree
  - Operation Mutator, Subtree mutator

Genetic Programs are basically just a type of Genetic Algorithm

A genetic program is just a tree with nodes

(Think LISP)

Nodes can be operators (functions) or terminals (accepting no inputs)

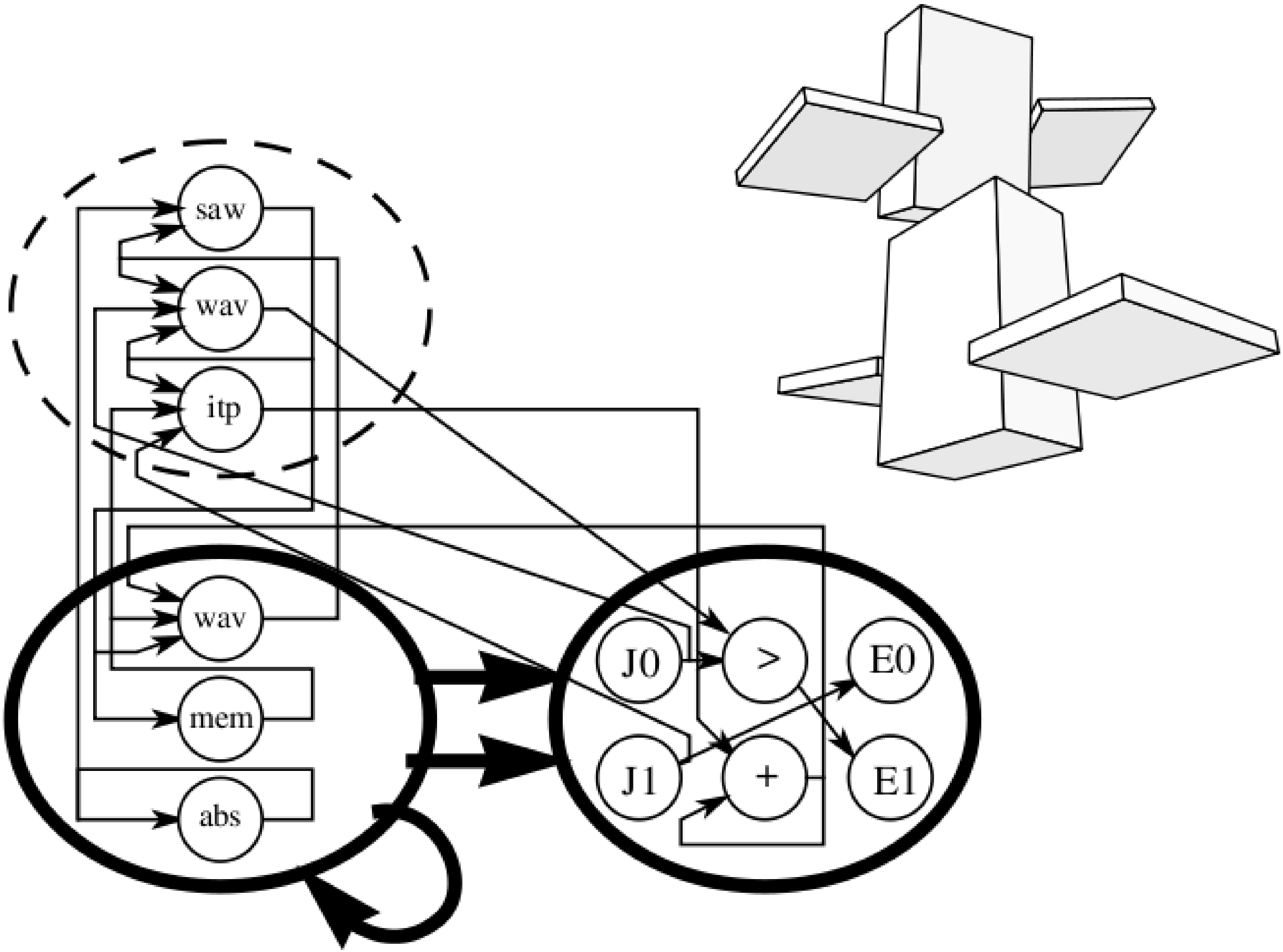

+
−
*
2.2
+
7
cos
X
11
Y

# Simple Operator Node Examples

- Addition, Subtraction, Multiplication, Division
  - Two inputs
- Cosine, Sine, Absolute Value
  - One input
- Max, Min, Average
  - Two inputs (or more)
- If Greater Than, If Less Than
  - Four inputs: if A is greater than B then C else D

# Terminal Node Examples

- Ephemeral Constants
  - Generated at runtime
- Specified Constants
  - Defined by you
- Variables
  - Value determined when evaluated


saw
wav
itp
wav
mem
abs
J0
>
E0
J1
+
E1

Putting it all together:

Creating a better unskilled forecast

# Defining The Problem

- Columbus Next-Day High Temperatures
  - From February 2005 to June 2010
- Average Absolute Error
  - Today's High: 5.74 degrees
  - Climate Average: 7.67 degrees
  - Yesterday's High: 8.01 degrees
  - Last Year's High: 10.87 degrees
- Can we beat that?

# Create The Genome

```
genotype = GTree.GTreeGP()
genotype.setParams(max_depth=4, method="ramped")
genotype.evaluator.set(eval_func)
```

# Create The Genome

```
genotype = GTree.GTreeGP()
genotype.setParams(max_depth=4, method="ramped")
genotype.evaluator.set(eval_func)
```

# Create The Genome

```
genotype = GTree.GTreeGP()
genotype.setParams(max_depth=4, method="ramped")
genotype.evaluator.set(eval_func)
```

# Define The Operators

```
@GTree.gpdec(representation="+", color="red")
def gp_add(a, b): return a+b

@GTree.gpdec(representation="-", color="red")
def gp_sub(a, b): return a-b

@GTree.gpdec(representation="avg", color="green")
def gp_avg(a, b): return (a+b)/2.0

@GTree.gpdec(representation="if_gt", color="blue")
def gp_gt(a, b, c, d):
    if a>b:
        return c
    else:
        return d
```

# Define The Operators

```
@GTree.gpdec(representation="+", color="red")
def gp_add(a, b): return a+b

@GTree.gpdec(representation="-", color="red")
def gp_sub(a, b): return a-b

@GTree.gpdec(representation="avg", color="green")
def gp_avg(a, b): return (a+b)/2.0

@GTree.gpdec(representation="if_gt", color="blue")
def gp_gt(a, b, c, d):
    if a>b:
        return c
    else:
        return d
```

# Define The Fitness Function

```
def eval_func(genome):
    code_comp = genome.getCompiledCode()

    error = 0.0
    count = 0.0
    for day, T, Y, L, C, actual in data:
        forecast = eval(code_comp)
        error += abs(forecast-actual)
        count += 1

    return error/count
```

# Define The Fitness Function

```
def eval_func(genome):
    code_comp = genome.getCompiledCode()

    error = 0.0
    count = 0.0
    for day, T, Y, L, C, actual in data:
        forecast = eval(code_comp)
        error += abs(forecast-actual)
        count += 1

    return error/count
```

# Define The Fitness Function

```
def eval_func(genome):
    code_comp = genome.getCompiledCode()

    error = 0.0
    count = 0.0
    for day, T, Y, L, C, actual in data:
        forecast = eval(code_comp)
        error += abs(forecast-actual)
        count += 1

    return error/count
```

# Initialize The Engine

```
ga = GSimpleGA.GSimpleGA(genotype)
```

```
ga.setParams(gp_terminals = ['T', 'L', 'C'],
             gp_function_prefix = "gp")
```

```
ga.setMinimax(Consts.minimaxType["minimize"])
ga.setGenerations(50)
ga.setCrossoverRate(1.0)
ga.setMutationRate(0.25)
ga.setPopulationSize(800)
```

# Initialize The Engine

```
ga = GSimpleGA.GSimpleGA(genotype)

ga.setParams(gp_terminals = ['T', 'L', 'C'],
             gp_function_prefix = "gp")

ga.setMinimax(Consts.minimaxType["minimize"])
ga.setGenerations(50)
ga.setCrossoverRate(1.0)
ga.setMutationRate(0.25)
ga.setPopulationSize(800)
```

# Initialize The Engine

```
ga = GSimpleGA.GSimpleGA(genotype)

ga.setParams(gp_terminals = ['T', 'L', 'C'],
             gp_function_prefix = "gp")

ga.setMinimax(Consts.minimaxType["minimize"])
ga.setGenerations(50)
ga.setCrossoverRate(1.0)
ga.setMutationRate(0.25)
ga.setPopulationSize(800)
```

# Initialize The Engine

```
ga = GSimpleGA.GSimpleGA(genotype)

ga.setParams(gp_terminals = ['T', 'L', 'C'],
             gp_function_prefix = "gp")
```

```
ga.setMinimax(Consts.minimaxType["minimize"])
ga.setGenerations(50)
ga.setCrossoverRate(1.0)
ga.setMutationRate(0.25)
ga.setPopulationSize(800)
```

# Run

```
ga = GSimpleGA.GSimpleGA(genotype)
ga.setParams(gp_terminals = ['per','perlast','clm'],
             gp_function_prefix = "gp")

ga.setMinimax(Consts.minimaxType["minimize"])
ga.setGenerations(50)
ga.setCrossoverRate(1.0)
ga.setMutationRate(0.25)
ga.setPopulationSize(800)

ga.evolve(freq_stats=10)
best = ga.bestIndividual()
print best
```

# Results

```
(pyenv)efloehr@cassini:~/dev/evolve-wx/pycon$ python step1.py

Gen. 0 (0.00%): Max/Min/Avg Fitness(Raw [108.74(375.30)/85.25(6.39)/90.61(90.61)]
Gen. 10 (10.00%): Max/Min/Avg Fitness(Raw) [20.13(249.98)/16.61(5.74)/16.77(16.77)]
Gen. 20 (20.00%): Max/Min/Avg Fitness(Raw) [43.01(187.38)/34.40(5.44)/35.84(35.84)]
Gen. 30 (30.00%): Max/Min/Avg Fitness(Raw) [43.93(250.52)/35.54(5.44)/36.61(36.61)]
Gen. 40 (40.00%): Max/Min/Avg Fitness(Raw) [35.58(187.40)/28.74(5.44)/29.65(29.65)]
Gen. 50 (50.00%): Max/Min/Avg Fitness(Raw) [35.51(156.22)/28.46(5.44)/29.59(29.59)]
Gen. 60 (60.00%): Max/Min/Avg Fitness(Raw) [45.45(218.53)/36.51(5.44)/37.87(37.87)]
Gen. 70 (70.00%): Max/Min/Avg Fitness(Raw) [42.81(203.04)/34.39(5.44)/35.68(35.68)]
Gen. 80 (80.00%): Max/Min/Avg Fitness(Raw) [27.86(125.07)/22.41(5.44)/23.22(23.22)]
Gen. 90 (90.00%): Max/Min/Avg Fitness(Raw) [23.58(187.58)/19.32(5.44)/19.65(19.65)]
Gen. 100 (100.00%): Max/Min/Avg Fitness(Raw) [32.72(188.44)/26.53(5.44)/27.26(27.26)]
```

# Results

```
- GTree
   Height:        2
   Nodes:         5

GTreeNodeBase [Childs=2] - [gp_avg]
  GTreeNodeBase [Childs=2] - [gp_avg]
    GTreeNodeBase [Childs=0] - [C]
    GTreeNodeBase [Childs=0] - [T]
  GTreeNodeBase [Childs=0] - [T]
```

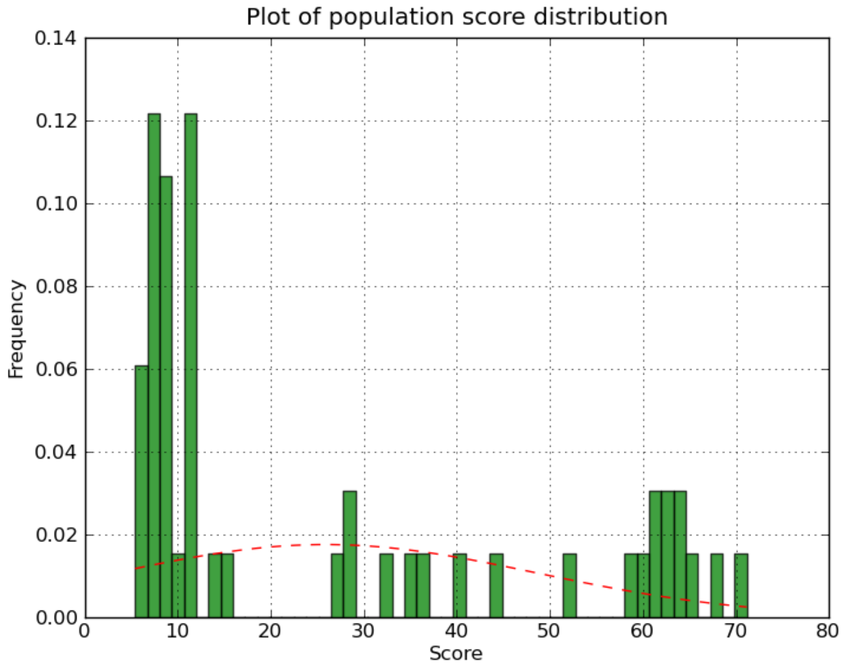
```
- GTreeGP
   Expression: gp_avg(gp_avg(C, T), T)
```

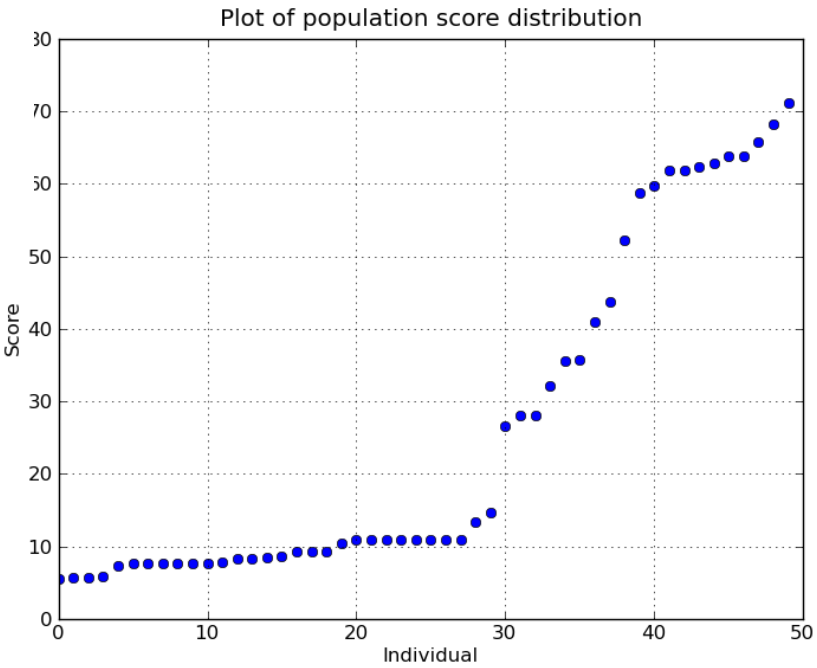
# The Shiny Stuff

# Interactive Mode

```
ga.setInteractiveGeneration(50)

>>> it.plotHistPopScore(population)

>>> it.plotPopScore(population)

>>> popScores = it.getPopScores(population)
```

# Interactive Mode

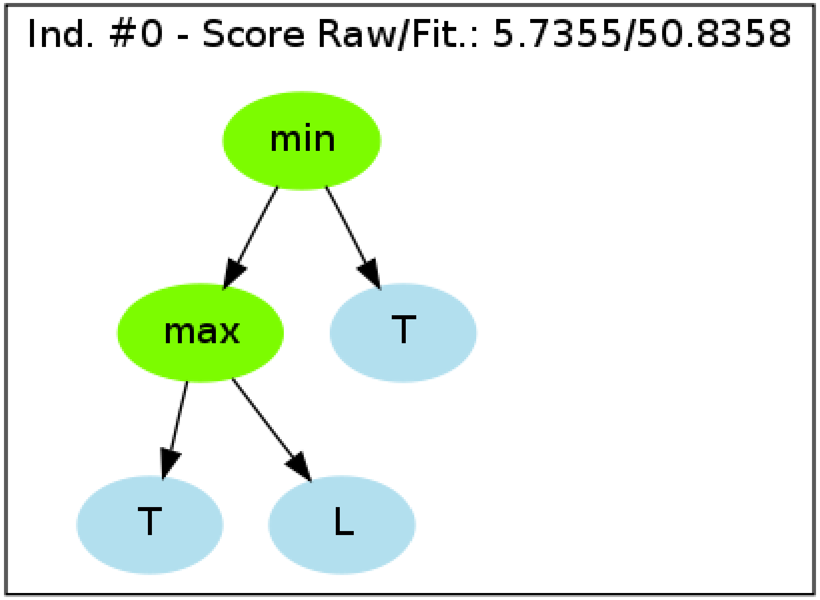


# GP Tree Graphs

```
Gtree.GTreeGP.writePopulationDot(gp, filename,
                                 "png", 0, 1)
```

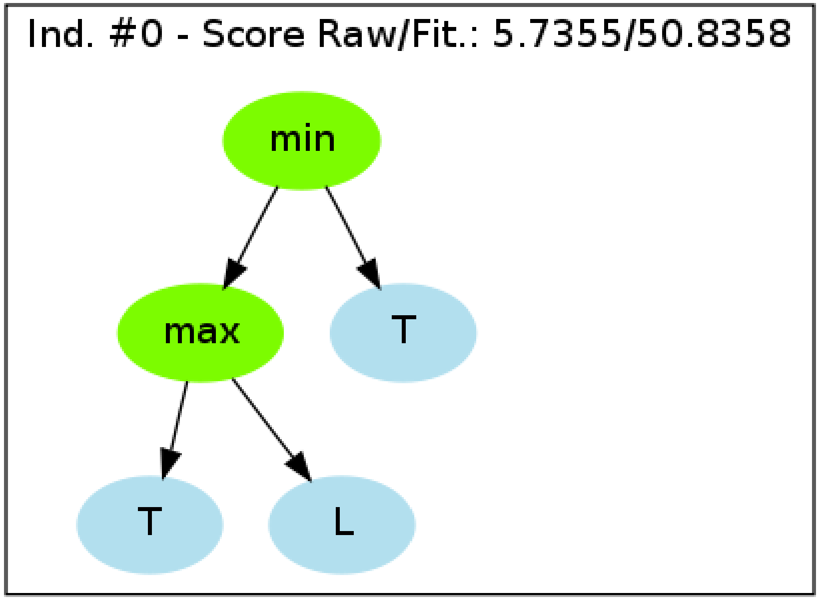
```
best = ga.bestIndividual()
best.writeDotImage("best.png")
```



# GP Tree Graphs

```
Gtree.GTreeGP.writePopulationDot(gp, filename,
                                  "png", 0, 1)
```

```
best = ga.bestIndividual()
best.writeDotImage("best.png")
```



# Callbacks

```
def step_callback(gp_engine):
    gen = gp_engine.getCurrentGeneration()
    if gen % 10 == 0:
        filename = "best_{gen}.png".format(gen=gen)
        GTree.GTreeGP.writePopulationDot(gp_engine,
                                         filename, "png", 0, 1)
```

```
ga.stepCallback.set(step_callback)
```

# Callbacks

```
def step_callback(gp_engine):
    gen = gp_engine.getCurrentGeneration()
    if gen % 10 == 0:
        filename = "best_{gen}.png".format(gen=gen)
        GTree.GTreeGP.writePopulationDot(gp_engine,
                                      filename, "png", 0, 1)
```

```
ga.stepCallback.set(step_callback)
```

# Database Adapters

```
csv_adapter = DBAdapters.DBFileCSV(identify="run1",
                                   filename="stats.csv")
ga.setDBAdapter(csv_adapter)
```

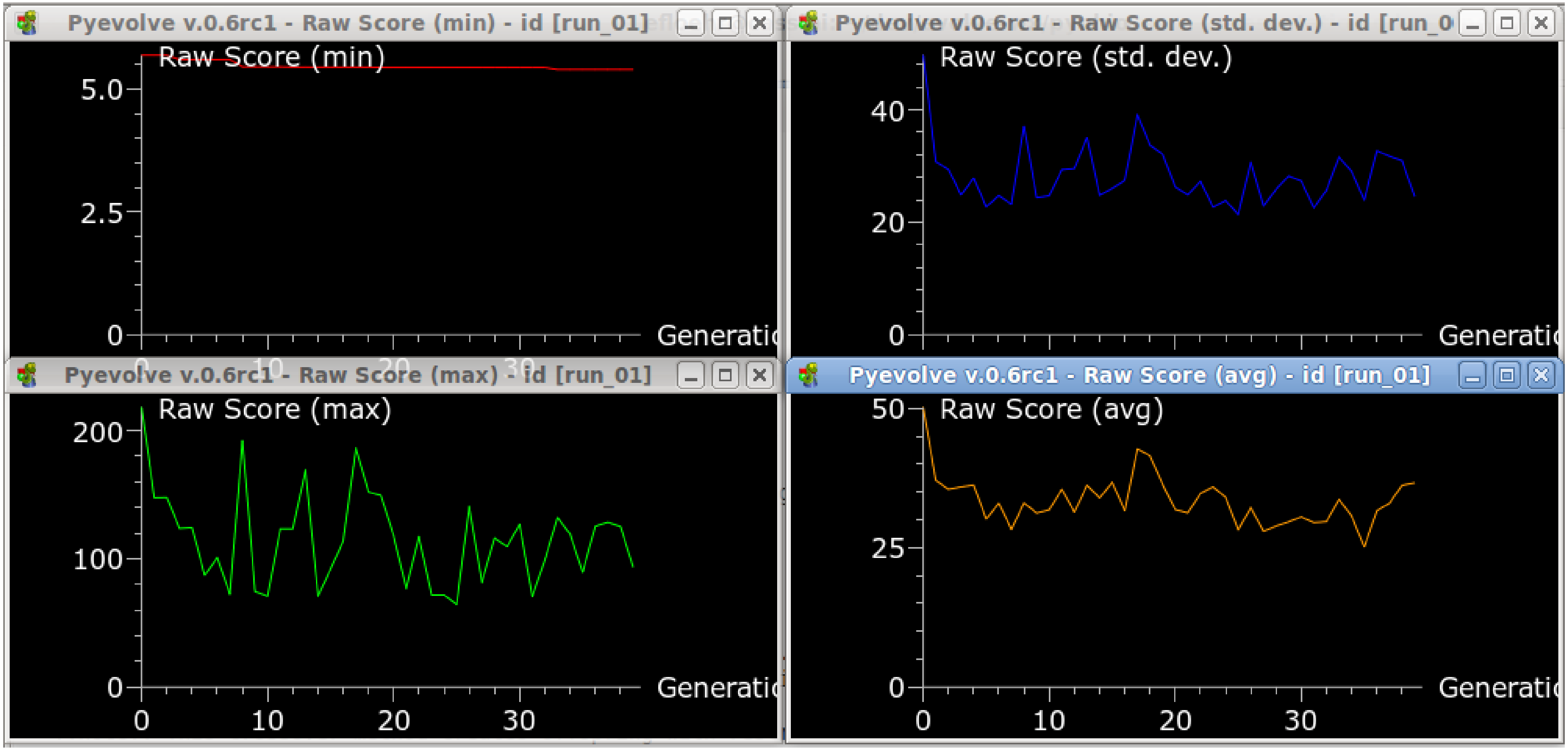
```
ga.evolve(freq_stats=10)

print ga.getStatistics()
```

```
- Statistics
    Minimum raw score                             = 7.35
    Fitness average                               = 16.43
    Minimum fitness                               = 16.32
    Raw scores variance                           = 1352.22
    Standard deviation of raw scores              = 36.77
    Average of raw scores                         = 16.43
    Maximum fitness                               = 19.72
    Maximum raw score                             = 295.57
```

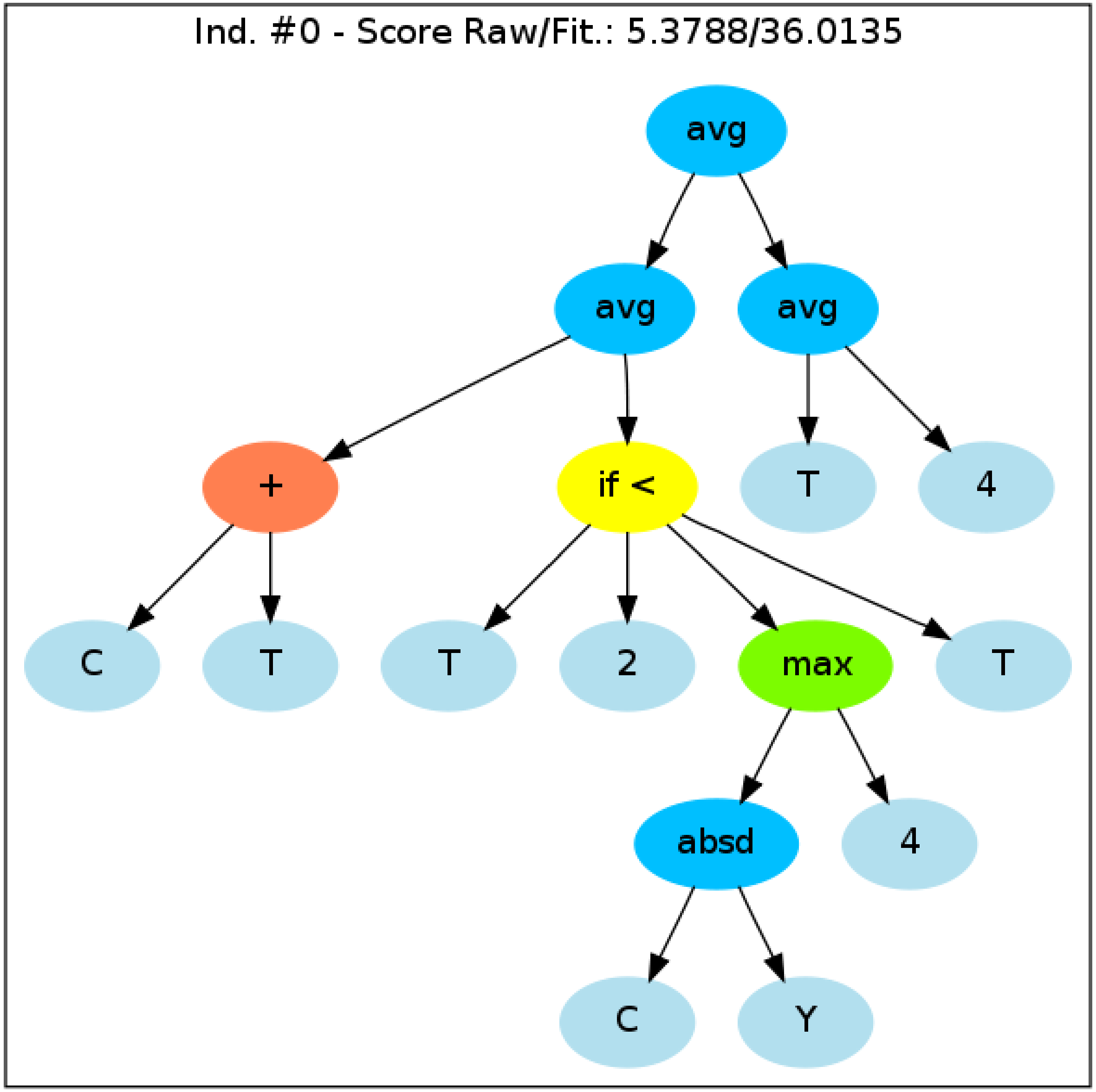
# Real-Time Plots

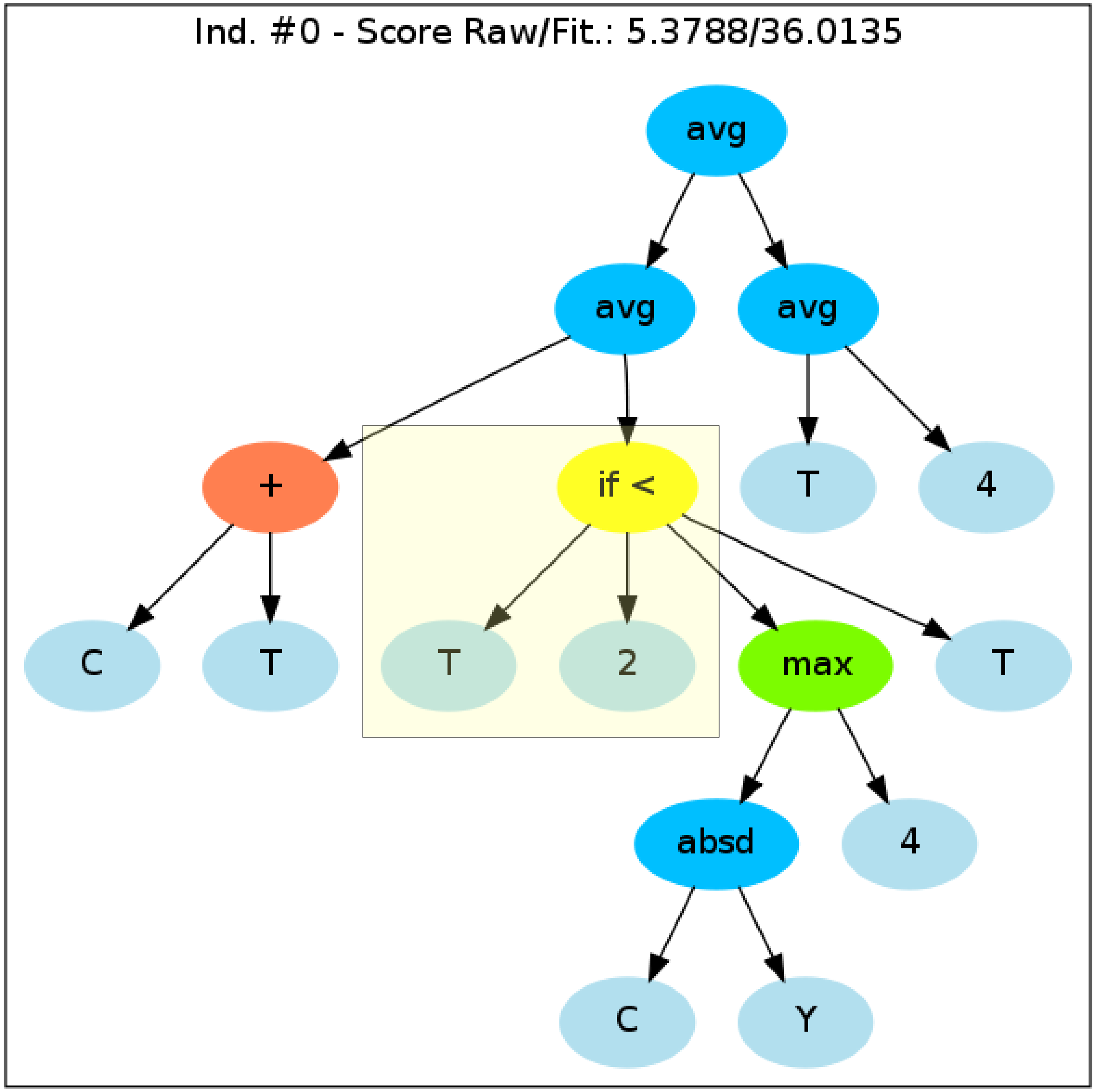
```
adapter = DBAdapters.DBVPythonGraph(identify="run_01",
                                    frequency = 1)

ga.setDBAdapter(adapter)
```

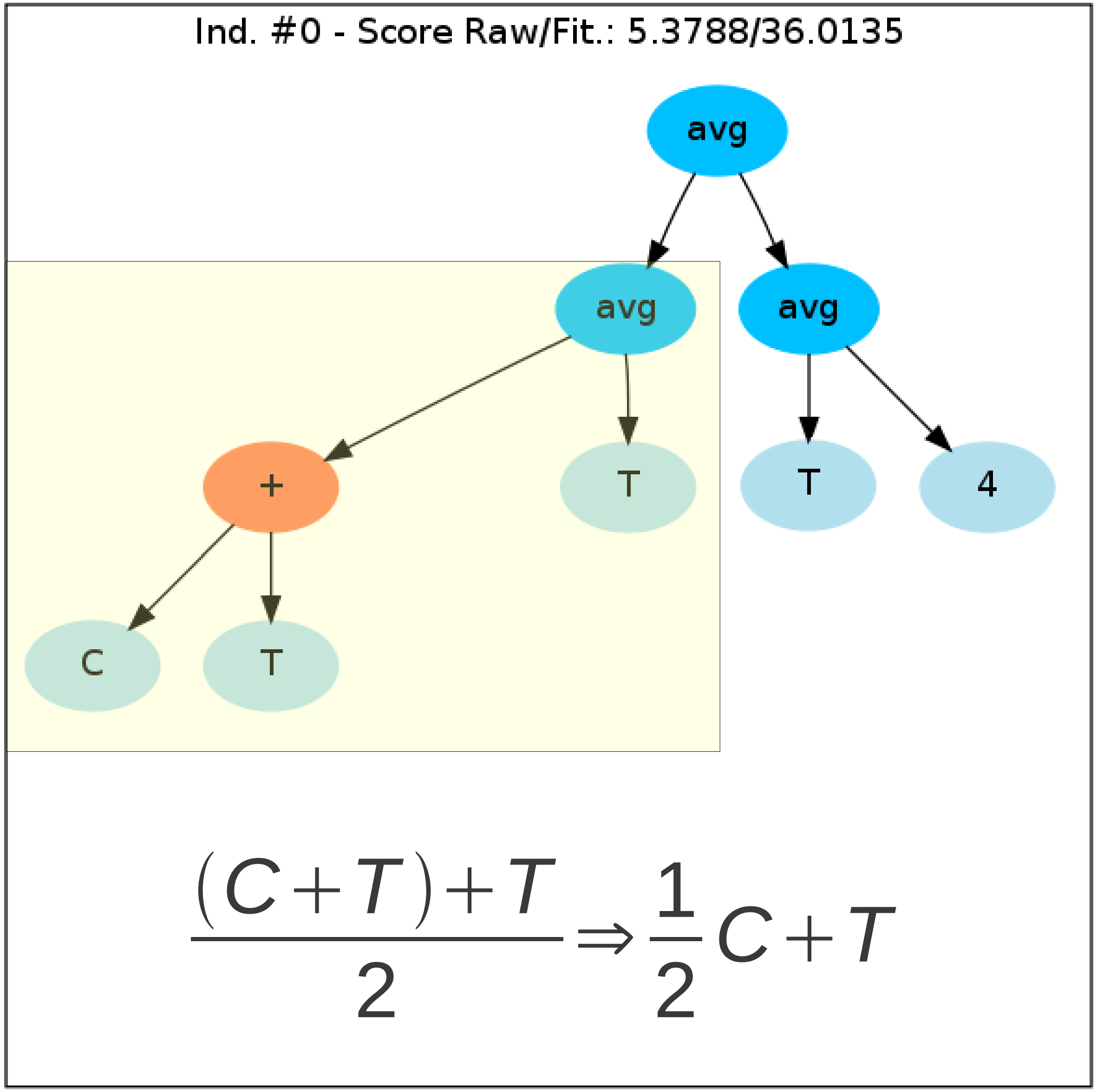
# Real-Time Plots

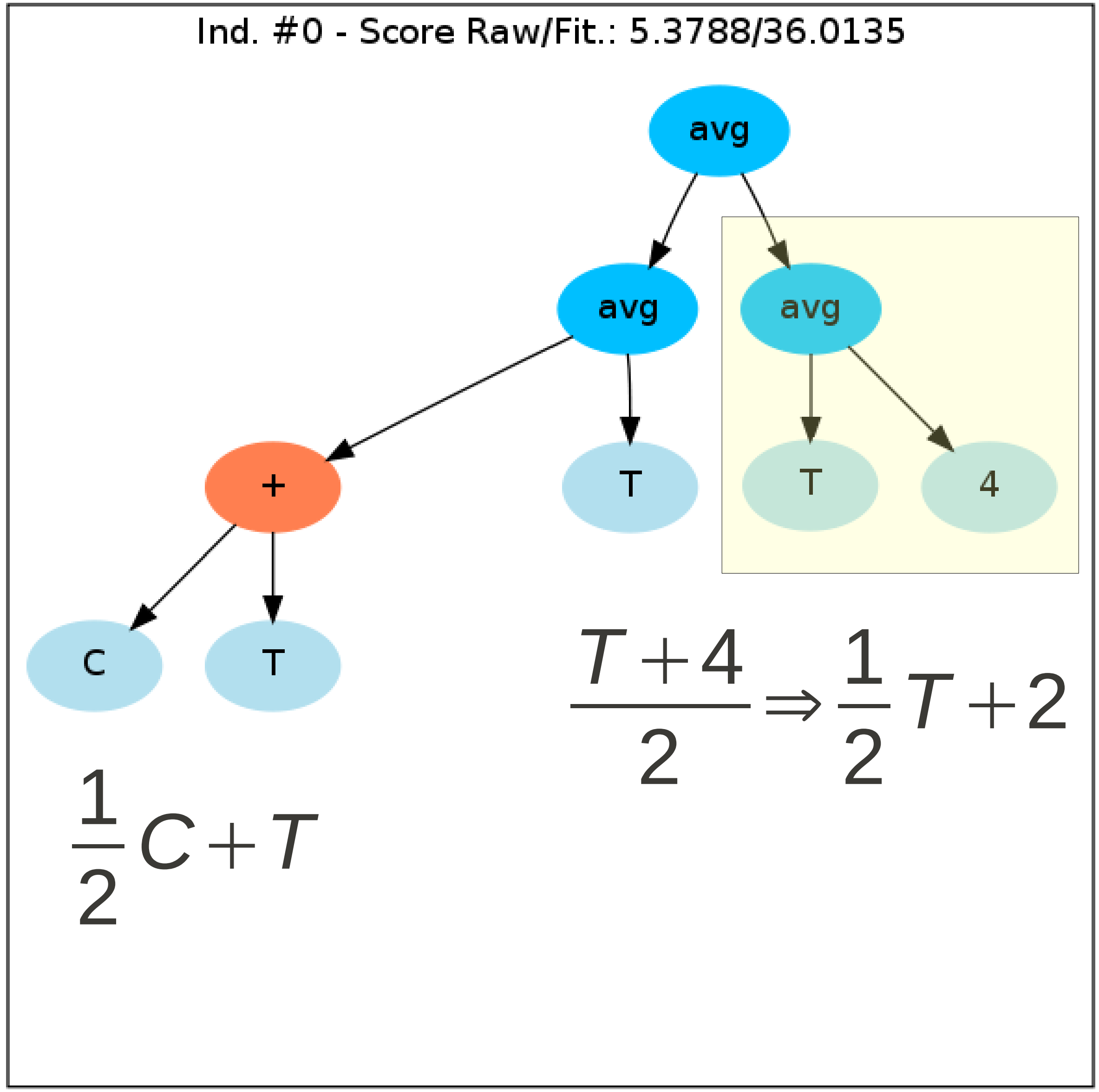


# Post-Processing Genomes

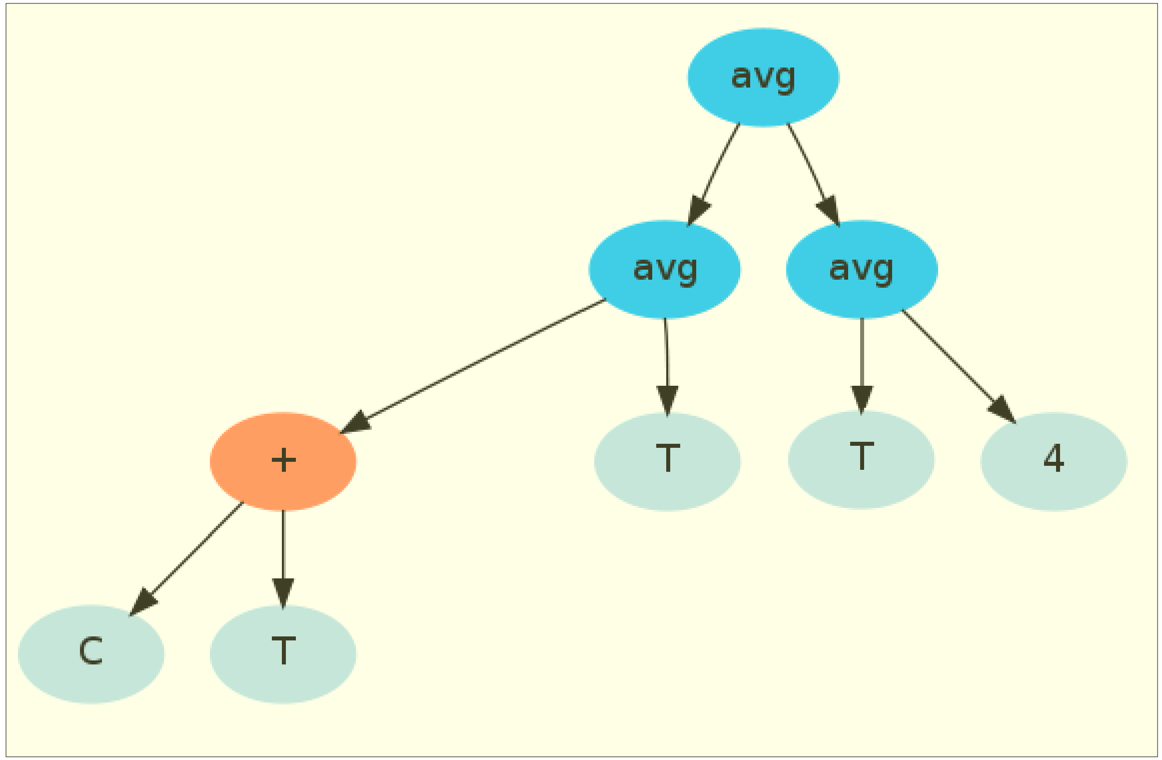

Ind. #0 - Score Raw/Fit.: 5.3788/36.0135
avg
avg
avg
+
if <
T
4
C
T
T
2
max
T
absd
4
C
Y


Ind. #0 - Score Raw/Fit.: 5.3788/36.0135
avg
avg
avg
+
if <
T
4
C
T
T
2
max
T
absd
4
C
Y

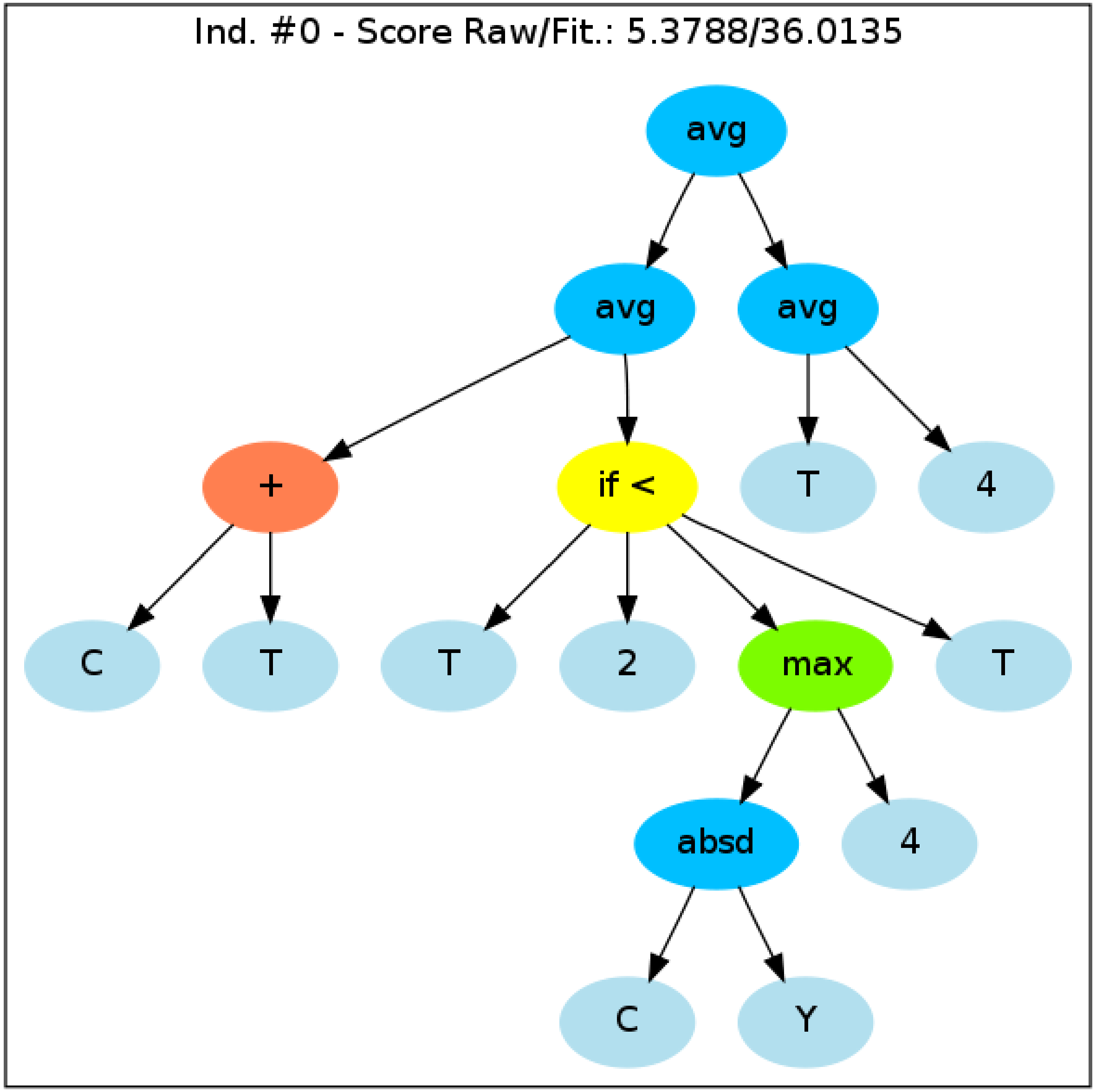

Ind. #0 - Score Raw/Fit.: 5.3788/36.0135
avg
avg
avg
+
T
T
4
C
T
$\frac{(C+T)+T}{2} \Rightarrow \frac{1}{2}C+T$


Ind. #0 - Score Raw/Fit.: 5.3788/36.0135
avg
avg
avg
+
T
T
4
C
T
$\frac{T+4}{2} \Rightarrow \frac{1}{2}T+2$
$\frac{1}{2}C+T$

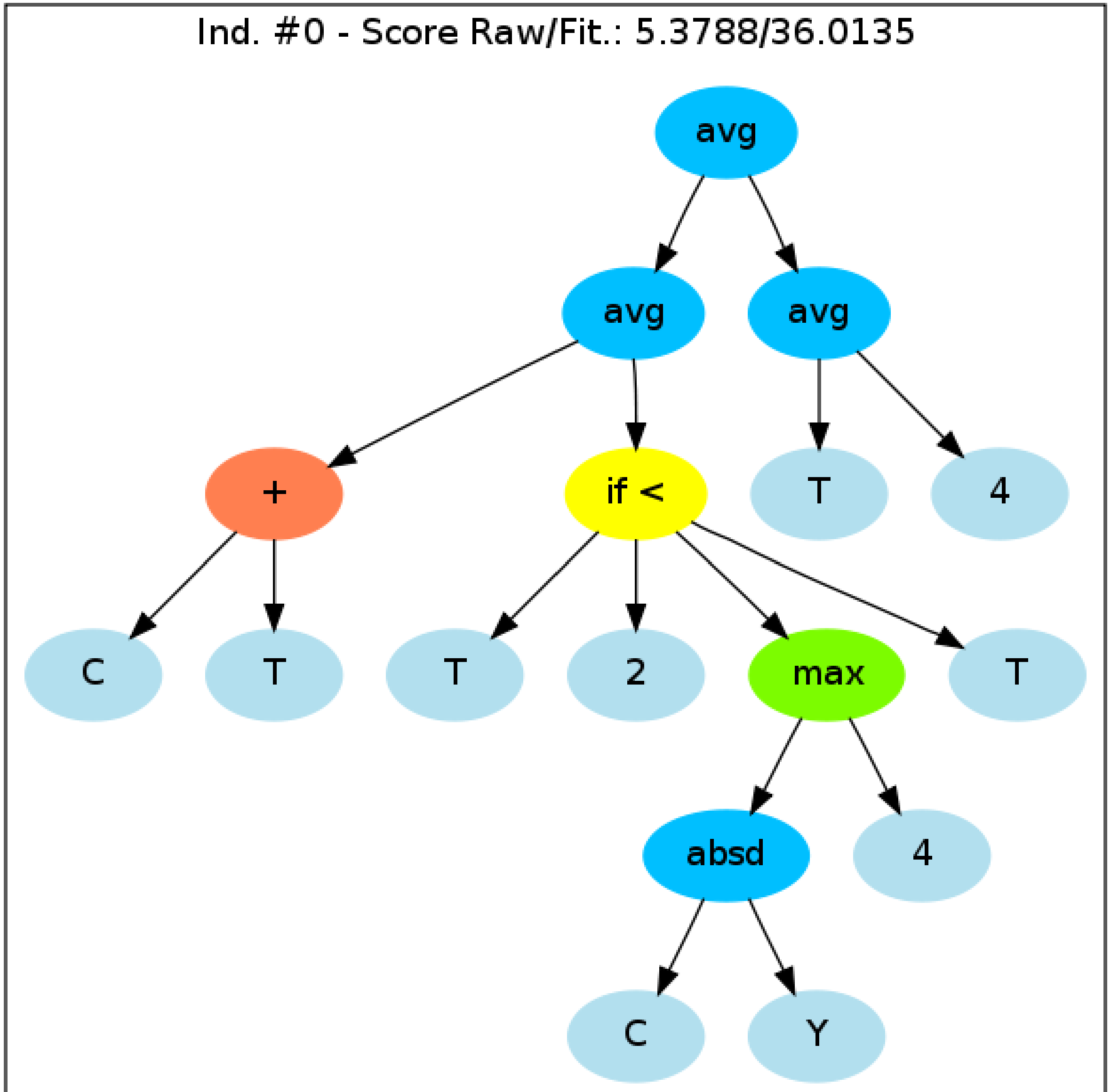
Ind. #0 - Score Raw/Fit.: 5.3788/36.0135



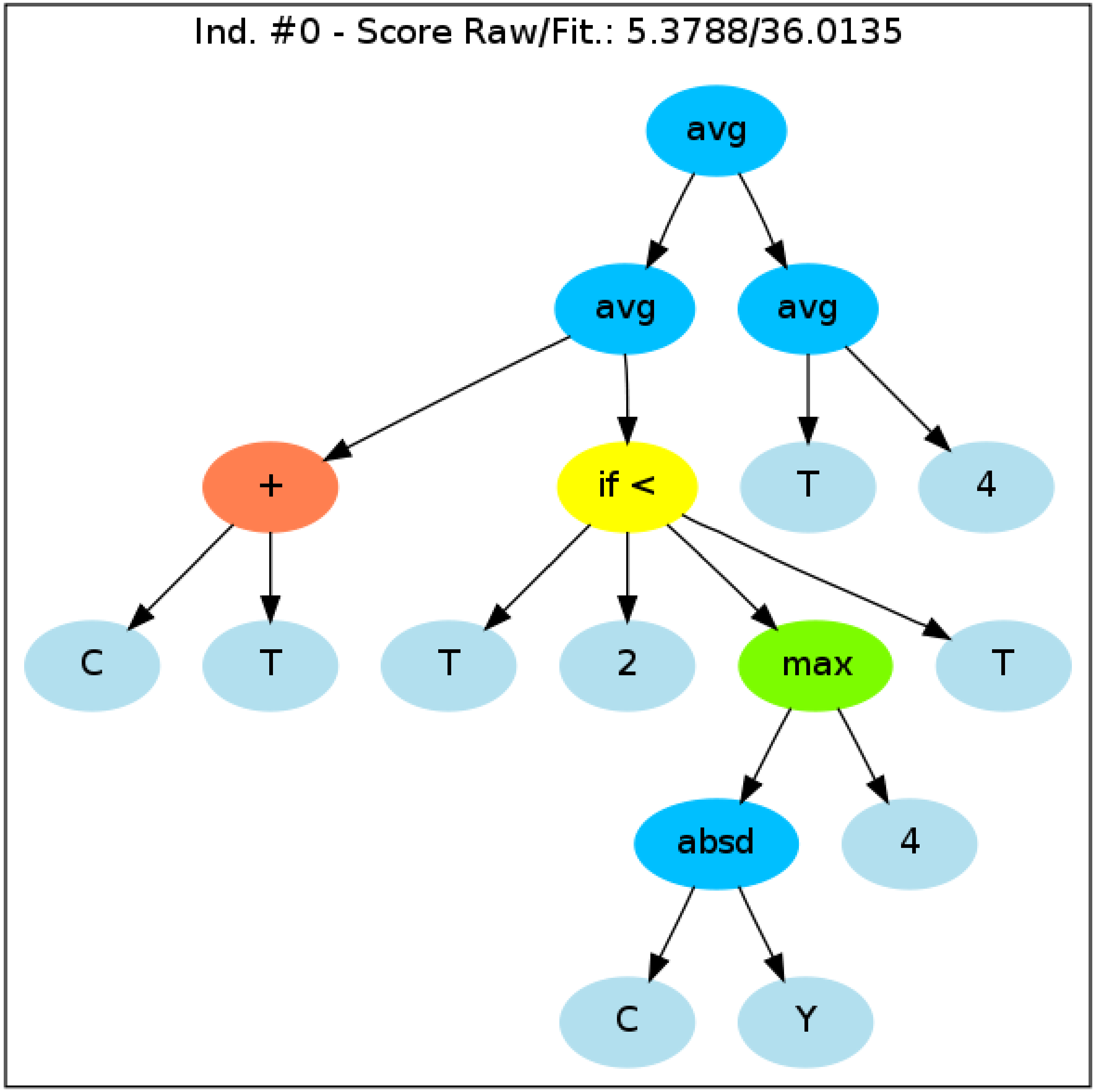

$$\frac{\frac{1}{2}C+T+\frac{1}{2}T+2}{2}$$



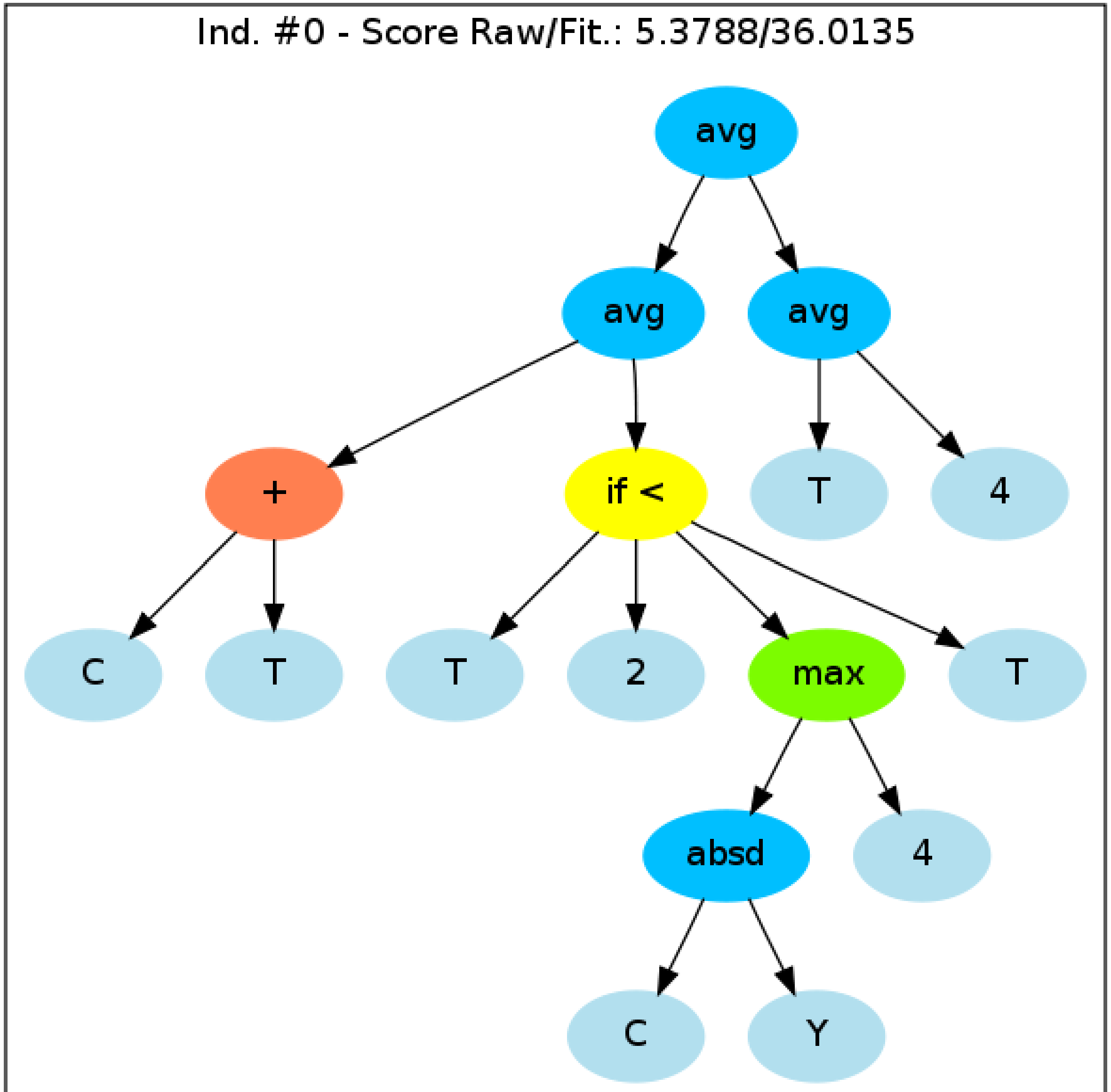

$$\frac{1}{4}C+\frac{3}{4}T+1$$



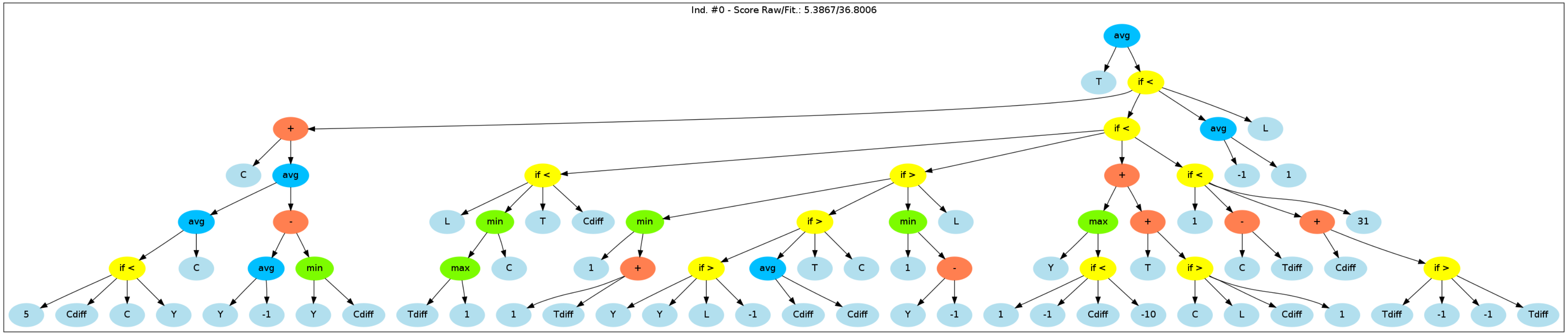

$$\boxed{\frac{1}{4}C}+\frac{3}{4}T+1$$




$$\frac{1}{4}C+\boxed{\frac{3}{4}T}+1$$




$$\frac{1}{4}C+\frac{3}{4}T\boxed{+1}$$


Ind. #0 - Score Raw/Fit.: 5.3867/36.8006
avg
T
if <
+
if <
avg
L
C
avg
if <
if >
+
if <
-1
1
avg
-
L
min
T
Cdiff
min
if >
min
L
max
+
1
-
+
31
if <
C
avg
min
max
C
1
+
if >
avg
T
C
1
-
Y
if <
T
if >
C
Tdiff
Cdiff
if >
5
Cdiff
C
Y
Y
-1
Y
Cdiff
Tdiff
1
1
Tdiff
Y
Y
L
-1
Cdiff
Cdiff
Y
-1
1
-1
Cdiff
-10
C
L
Cdiff
1
Tdiff
-1
-1
Tdiff

# Comparison

- Average absolute error:
  - Our GP: 5.38 degrees
  - Today's High: 5.74 degrees
  - Climate Average: 7.67 degrees
  - Yesterday's High: 8.01 degrees
  - Last Year's High: 10.87 degrees
- Improved on persistence forecast (today's high will be tomorrow's high) by 0.36 degrees.
- Average skilled forecast is about 3 degrees error

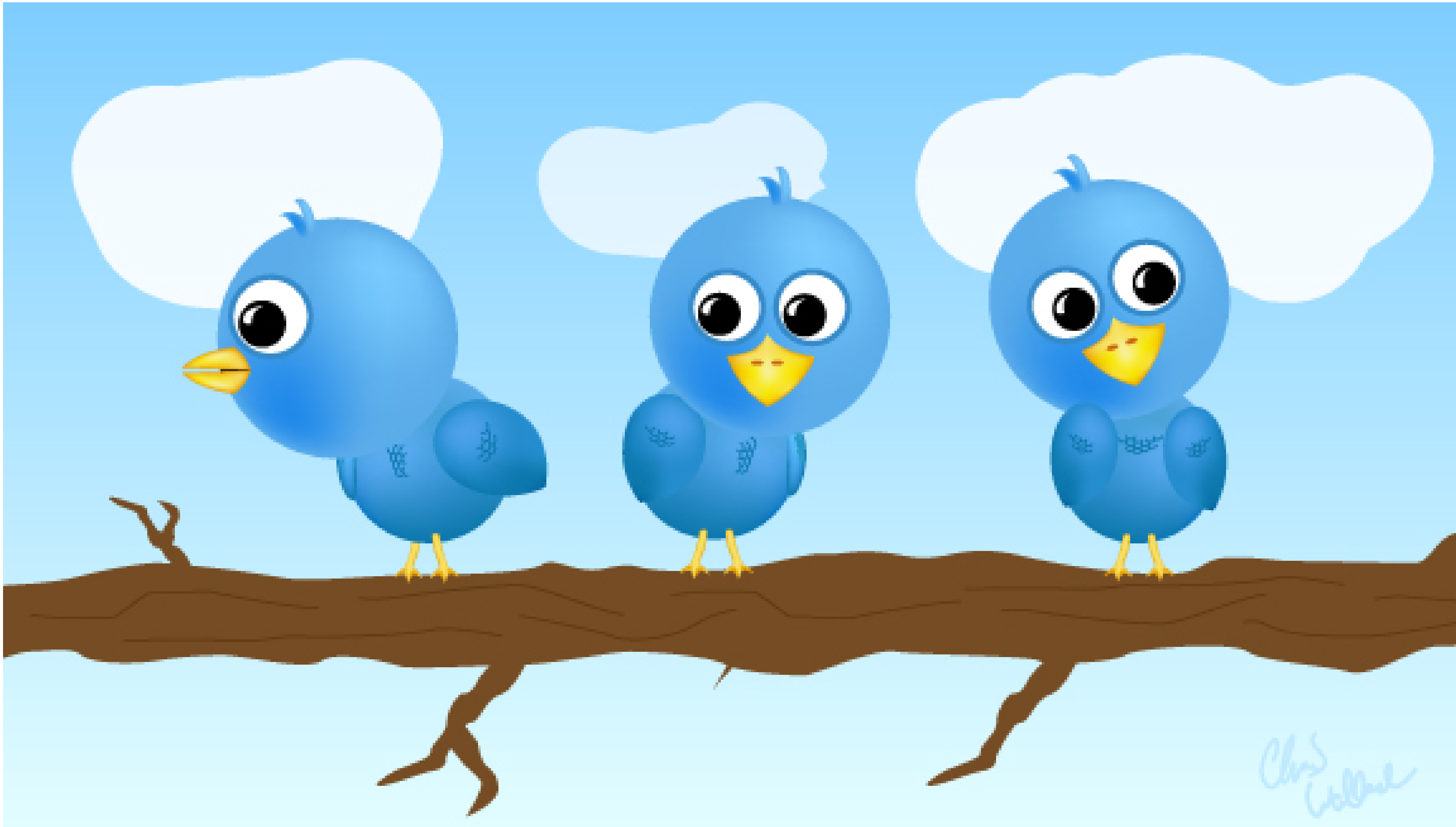
# The Challenge

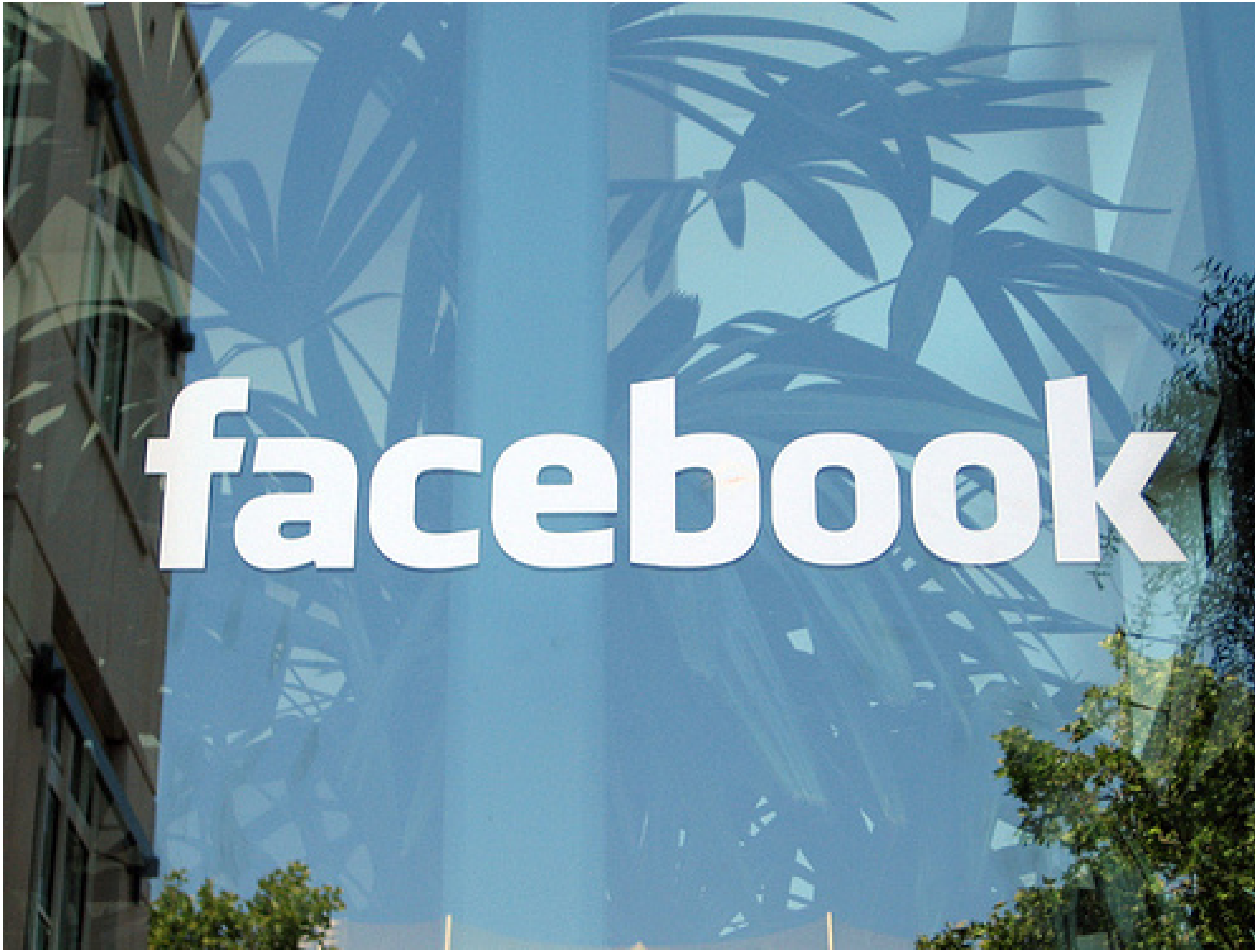
# There is a supercomputer under your desk

=

# There is a supercomputer under your desk

| | CM-5 | Dell T-3500 |
|---|---|---|
| *Year* | 1993 | 2010 |
| *CPUs* | 32 | 4 |
| *MHz* | 32 | 2933 |
| *Architecture* | SPARC | x86-64 |
| *Memory* | 1GB | 12GB |
| *MFLOPS* | 4,096 | 47,000 |
| *Cost* | $1,400,000 | $2,000 |

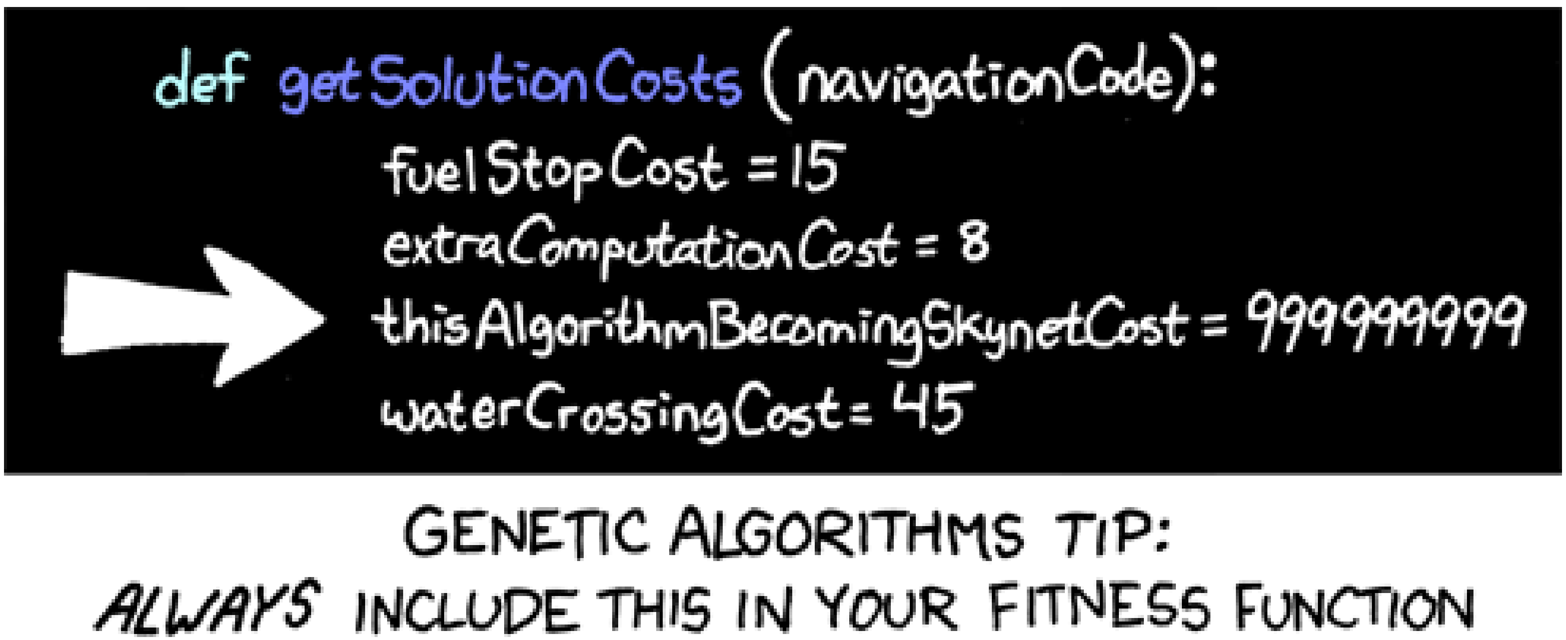



facebook


OMG! THEY THINKS I IZ
TINY CHEEZBURGER!

# Use the CPU!

# With This New Power Comes Responsibility




http://xkcd.com/534/

# EvoGuido

- Inspired by EvoLisa, used first picture returned by Google of BDFL (you can run with any pic)
- Hacked a little last night, needs work
- Will have time Sunday to work on (but hack away – framework there!)
- Improvements include allowing a varying number of polygons, better mutation and crossover ops (including polygon move, add, remove)

www.intellovations.com/pyevolve

http://is.gd/pyevolve    http://bit.ly/gHJhQ6

eric@intellovations.com

@ForecastWatch

Intellovations
Software for Discovery

Eric Floehr
Intellovations, LLC
eric@intellovations.com
(614) 440-0130