Linear Genetic Programming In Python Bytecode

Mark Burgess

ANU PhD student

markburgess1989@gmail.com
http://oddblog.info

Audience Poll

Put the hand up If:

A) you know what “Genetic Programming” is?

Audience Poll

Put the hand up If:

A) you know what “Genetic Programming” is?

B) have ‘done’ “Genetic Programming”?

Audience Poll

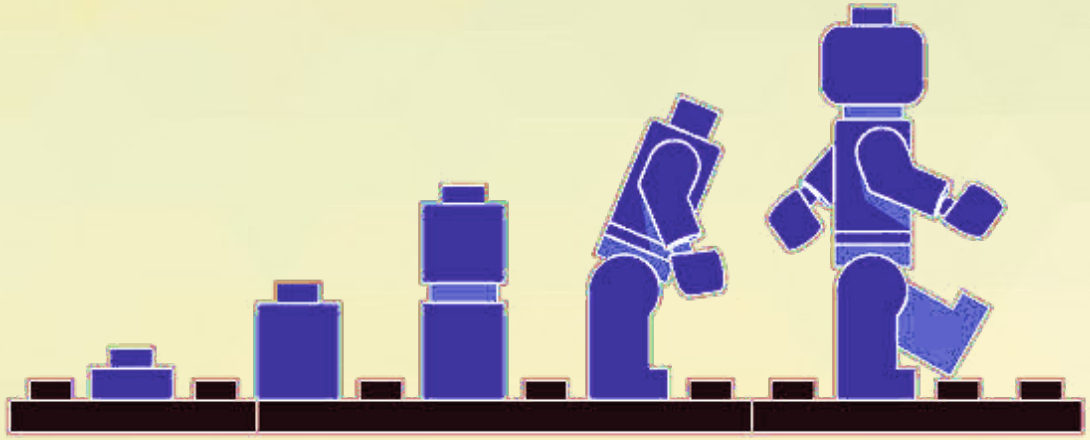
Put the hand up If:

A) you know what “Genetic Programming” is?

B) have ‘done’ “Genetic Programming”?

C) have fiddled with Python Bytecode?

Evolutionary Paradigm

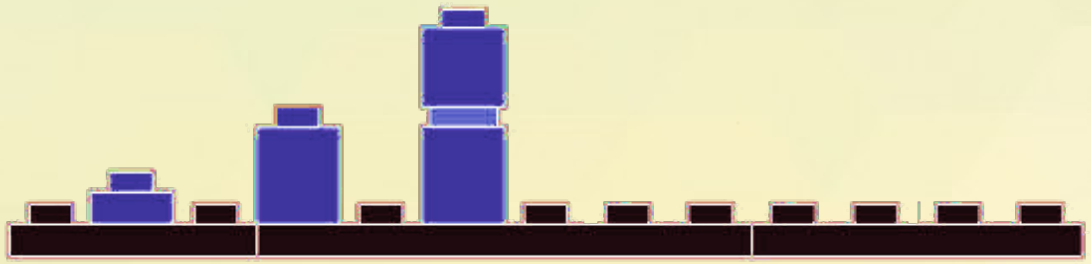


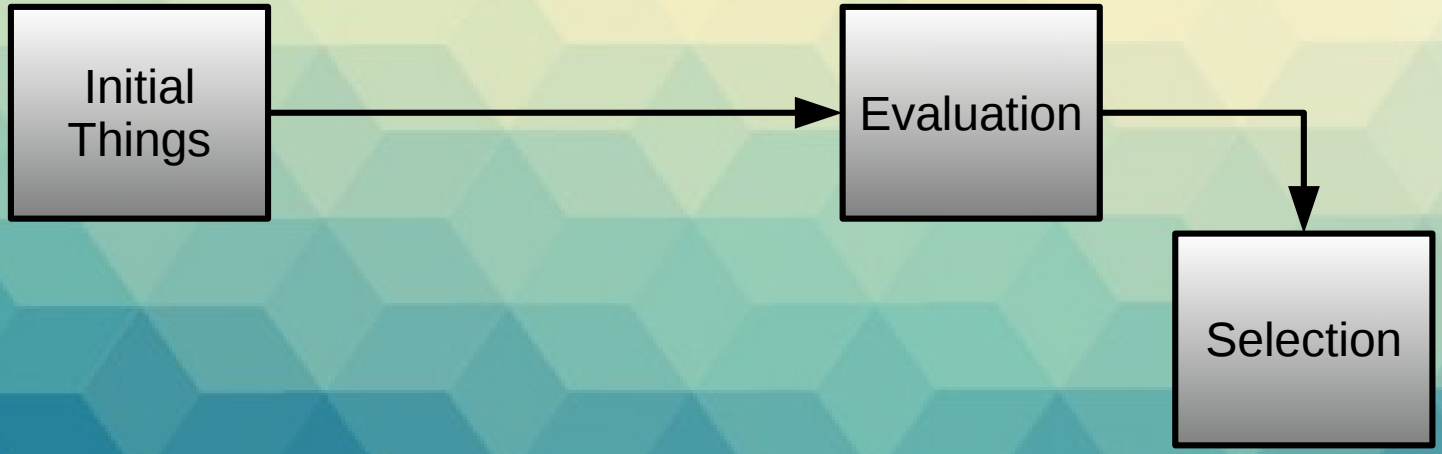
Evolutionary Paradigm

Initial Things

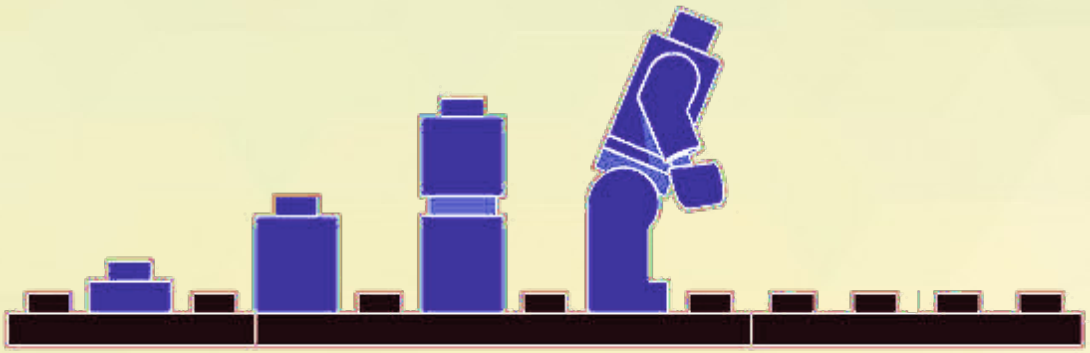
Evolutionary Paradigm

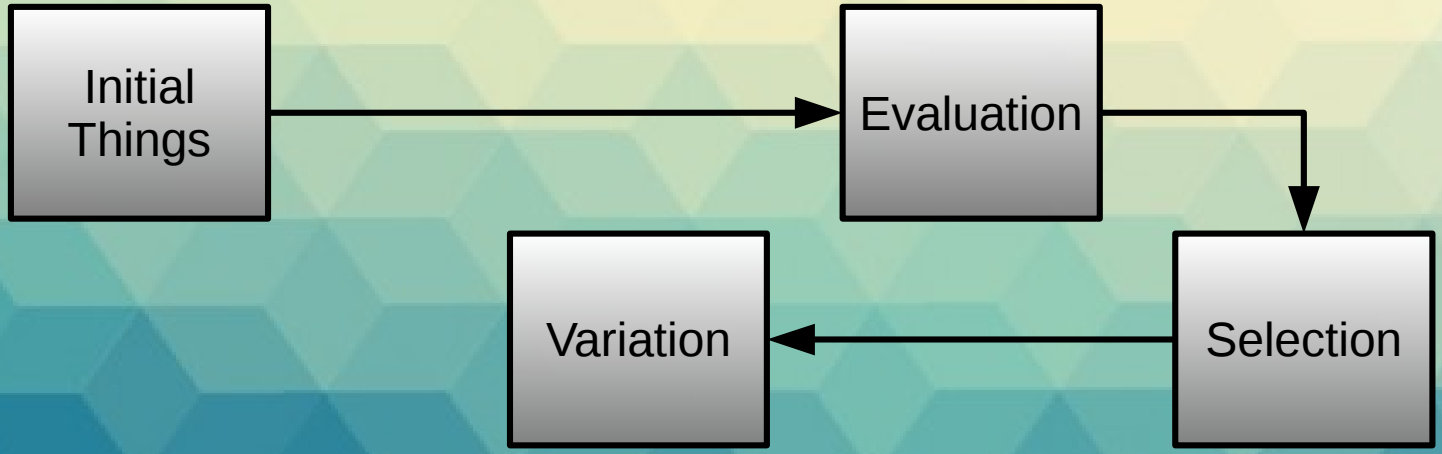
Evolutionary Paradigm



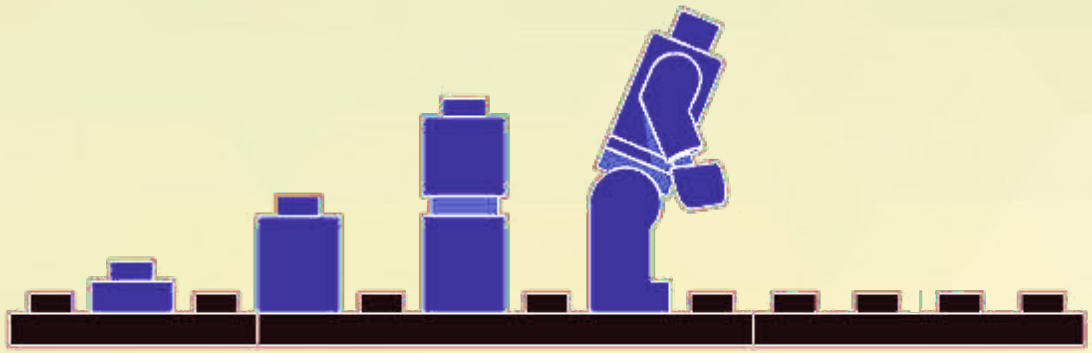


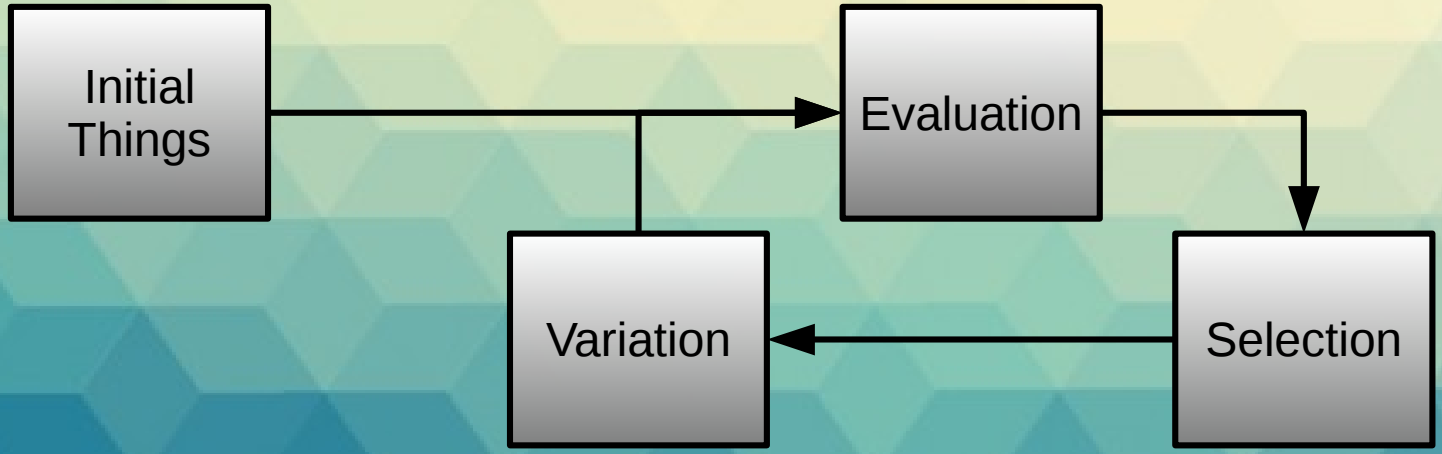
Evolutionary Paradigm



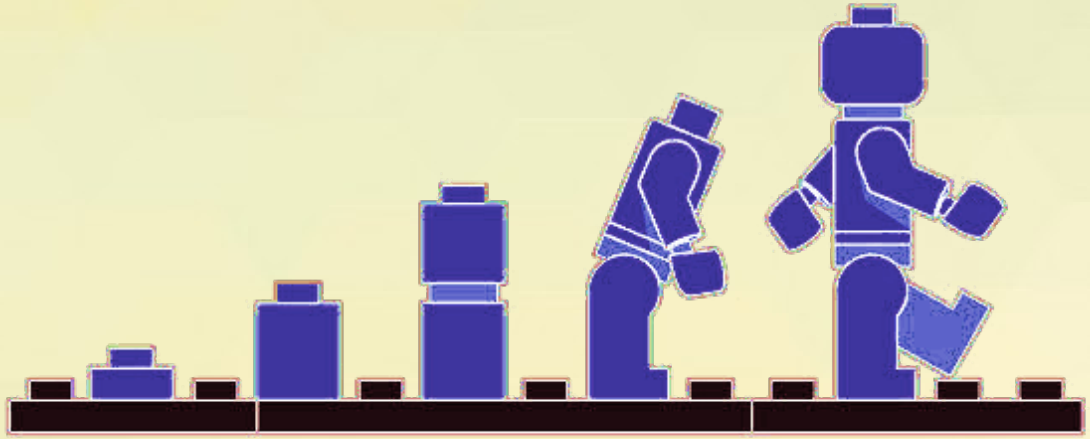


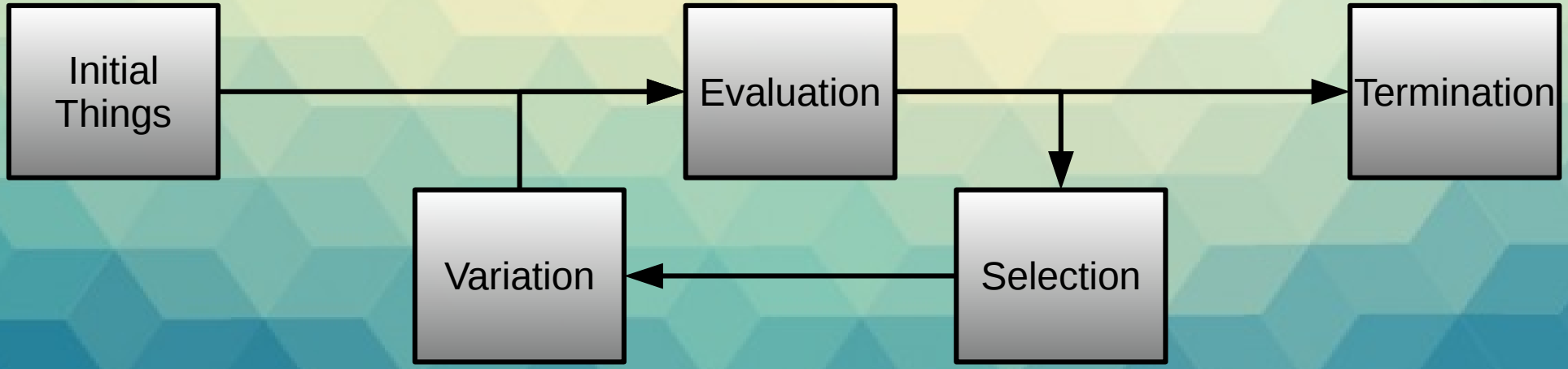
Evolutionary Paradigm



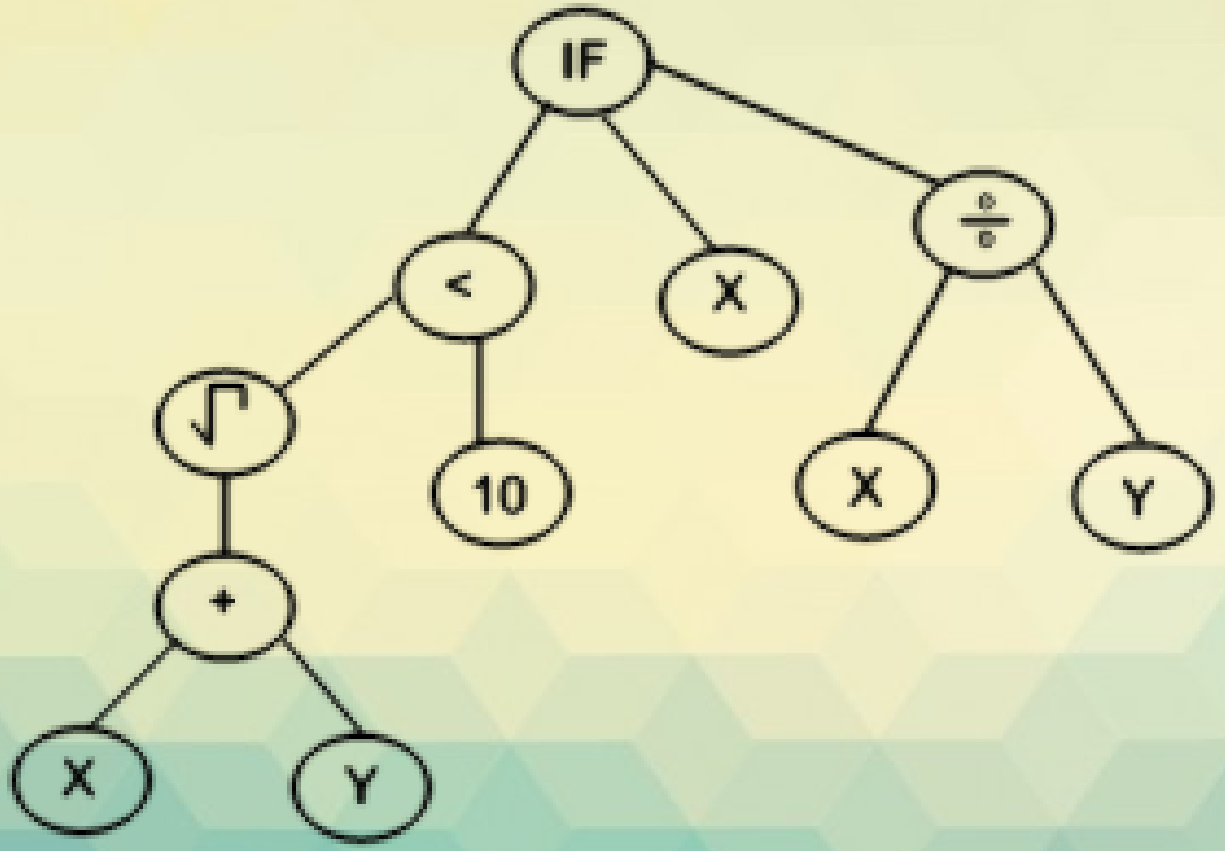


Evolutionary Paradigm





Organisms as Programs



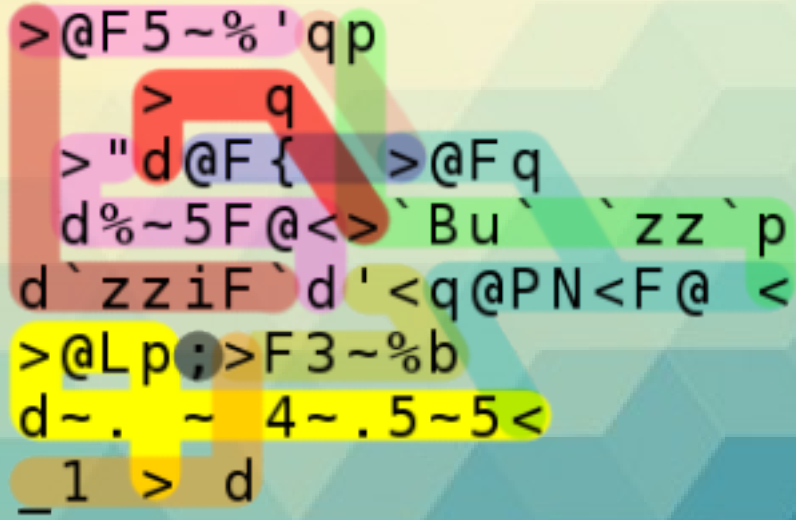
Organisms as Programs

```
 0 LOAD_FAST                0 (a)
 3 LOAD_FAST                1 (b)
 6 COMPARE_OP               4 (>)
 9 POP_JUMP_IF_FALSE       24

12 LOAD_FAST                1 (b)
15 LOAD_FAST                2 (c)
18 LOAD_FAST                0 (a)
21 BINARY_TRUE_DIVIDE
22 BINARY_ADD
23 RETURN_VALUE
```

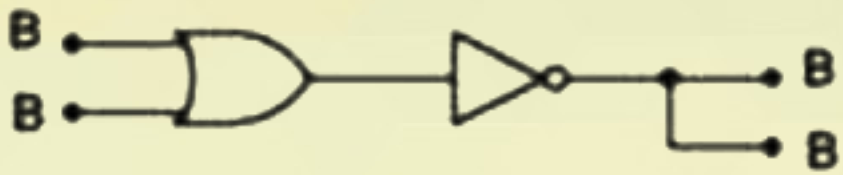
Organisms as Programs

```
++++++++++[>+++++++>++++++++++>+++<
<<-]>++++++.>+++++++++++++++.+++++++.----
-----.++++.[>>>>+< <<<-]>>>>[< <+<<+>>>>-]
< <<<-----.[>>>>+< <<<-]>>>>[< +<<<+>>>>-]
<  <<<-.-----.>>.--------------.<  <[>>>>+<  <<<-
]>++.< <-------.>>>>.+++++++.>----.
```

⟨S⟩→⟨cond$_I$⟩
⟨cond$_I$⟩→⟨cmp⟩
| **OR** ⟨cmp⟩ ⟨cond$_I$⟩
| **AND** ⟨cmp⟩ ⟨cond$_I$⟩
⟨cmp⟩→ ⟨op-num⟩ ⟨variable⟩ ⟨value⟩
| ⟨op-cat⟩ ⟨variable⟩ ⟨value⟩
⟨op-cat⟩ → **EQ| NOT EQ**
⟨op-num⟩→ **GT|GE| LT| LE**
⟨variable⟩ → *Any valid attribute in dataset*
⟨value⟩ → *Any valid value*



Organisms as Programs

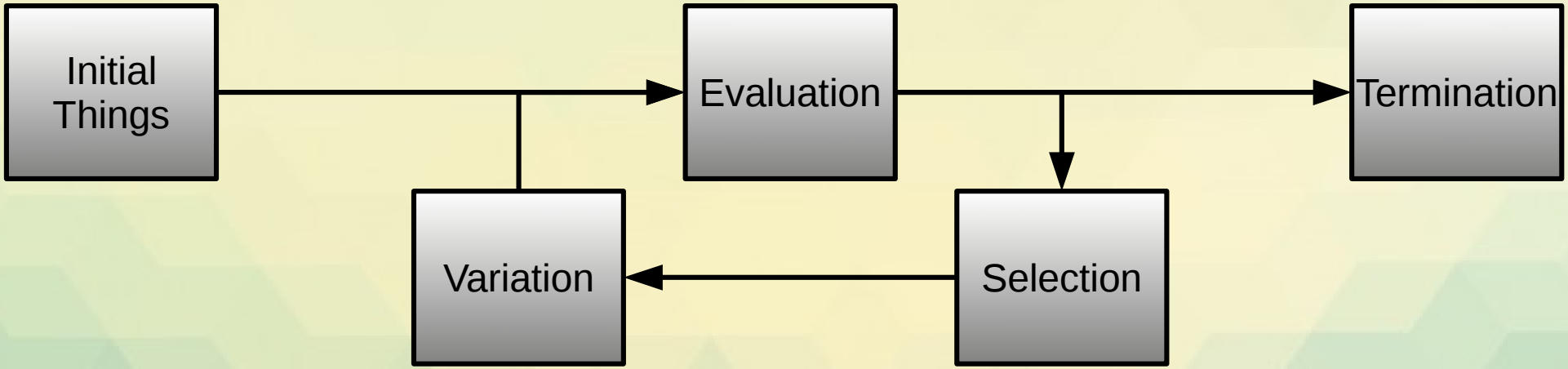


2B OR NOT 2B

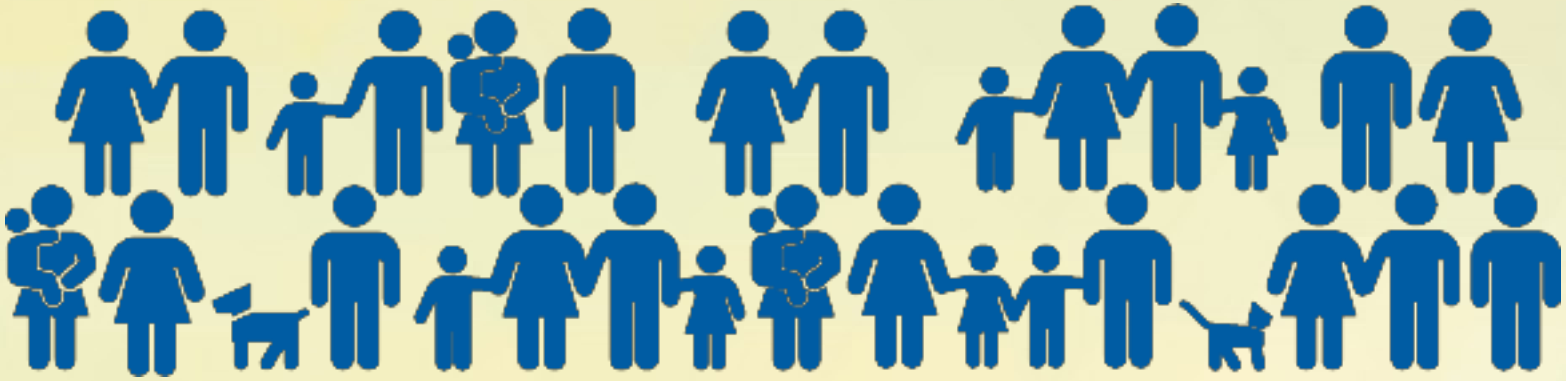
Choose/Design a Representation/Language

Type-consistency
Evaluation safety
‘Sufficiency’

Defines the Search Space


Initial Things
Evaluation
Termination
Variation
Selection

Initialisation

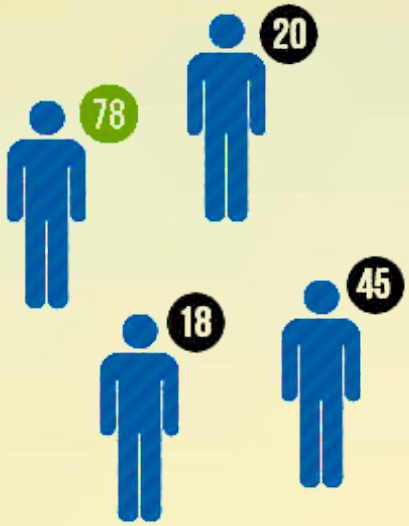


Initial population

Generated Algorithmically
(valid and varied)

Seeded

Evaluation



Usually according to a “Fitness Function”

Multi-facted
Comparative
Executing
Smooth

Selection

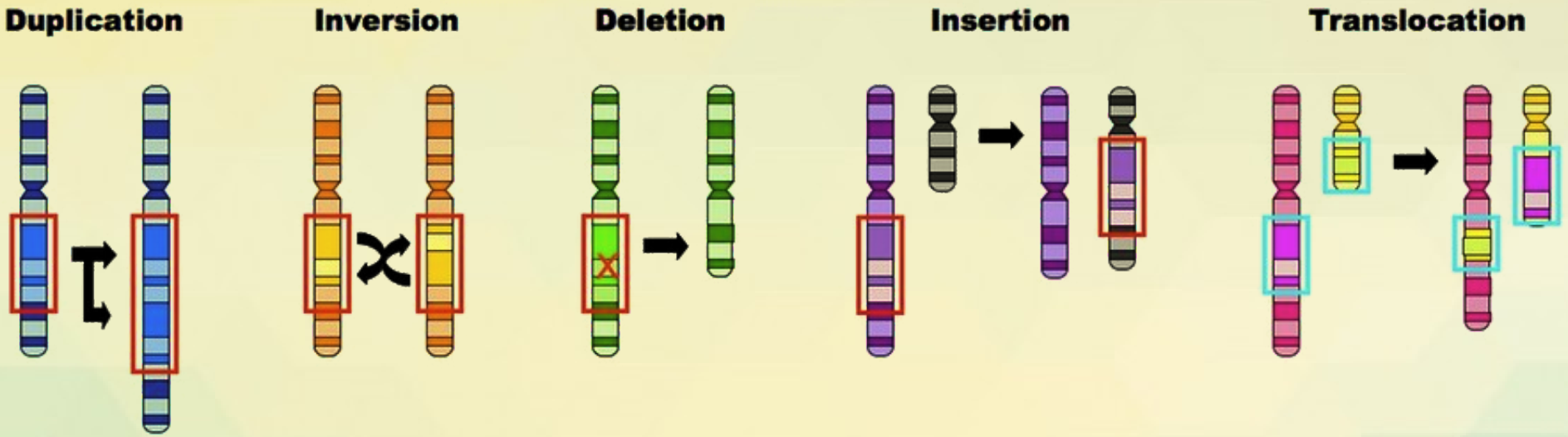
Techniques for broad select of ‘fitter’ individuals

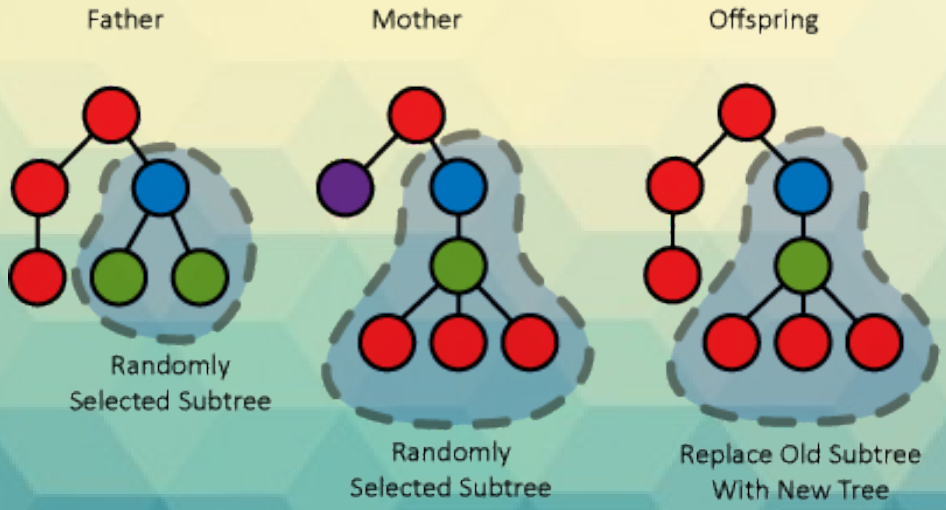
eg. Tournament – Roulette – Deterministic

Greediness

Evaluation as needed

Variation


Duplication
Inversion
Deletion
Insertion
Translocation

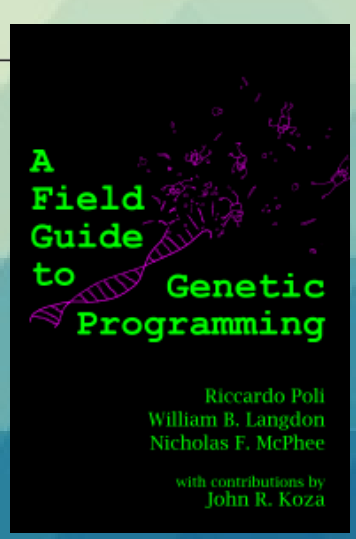

Father
Mother
Offspring
Randomly
Selected Subtree
Randomly
Selected Subtree
Replace Old Subtree
With New Tree

Iteration and Termination

An Example – Symbolic Regression

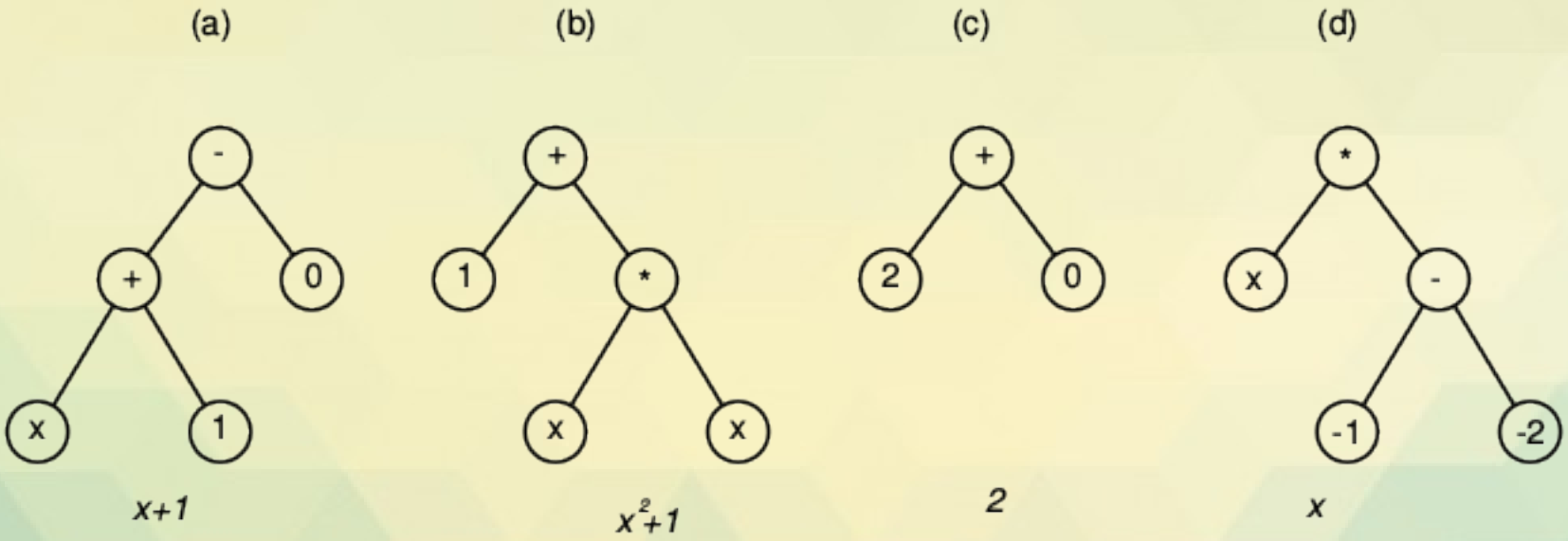
Table 4.1: Parameters for example genetic programming run

Objective:	Find program whose output matches $x^2 + x + 1$ over the range $-1 \leq x \leq +1$.
Function set:	$+$, $-$, % (protected division), and $\times$; all operating on floats
Terminal set:	x, and constants chosen randomly between -5 and $+5$
Fitness:	sum of absolute errors for $x \in \{-1.0, -0.9, \ldots 0.9, 1.0\}$
Selection:	fitness proportionate (roulette wheel) non elitist
Initial pop:	ramped half-and-half (depth 1 to 2. 50% of terminals are constants)
Parameters:	population size 4, 50% subtree crossover, 25% reproduction, 25% subtree mutation, no tree size limits
Termination:	Individual with fitness better than 0.1 found

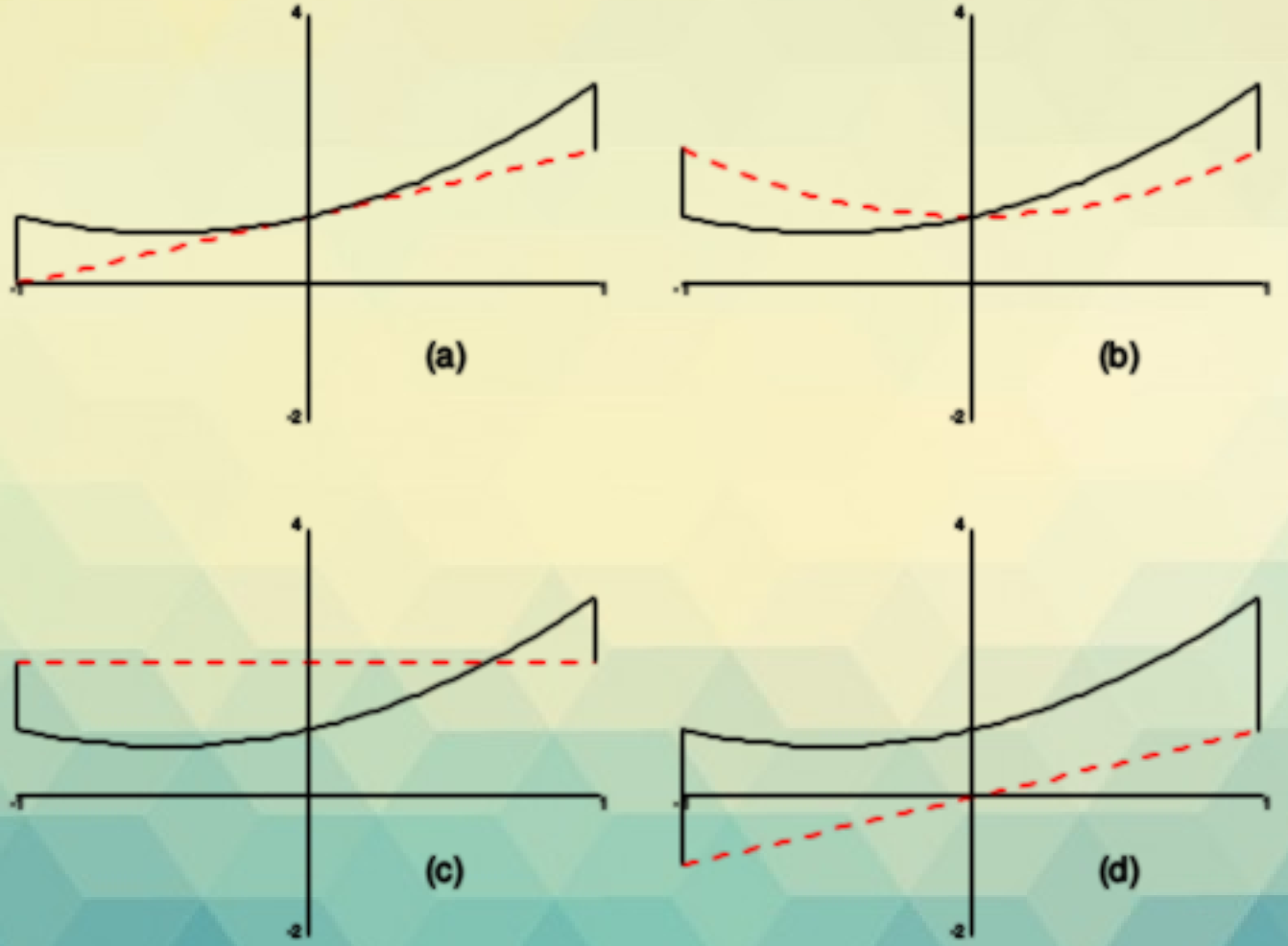


http://www.gp-field-guide.org.uk/

An Example – Symbolic Regression

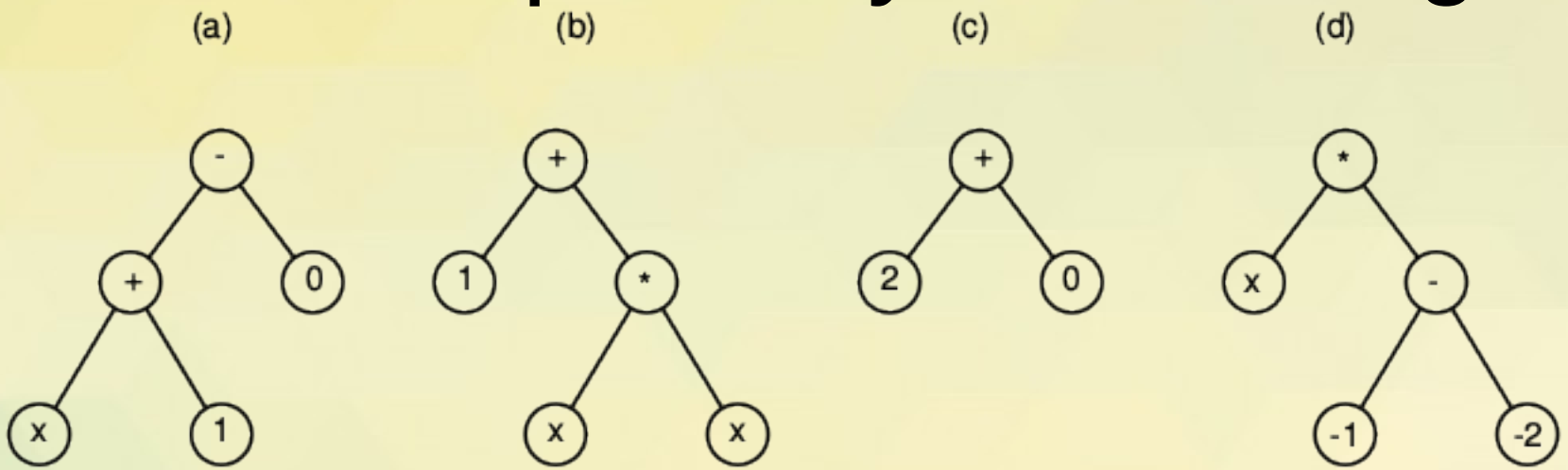


An Example – Symbolic Regression



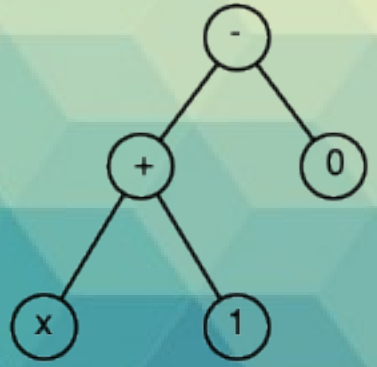
(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

An Example – Symbolic Regression



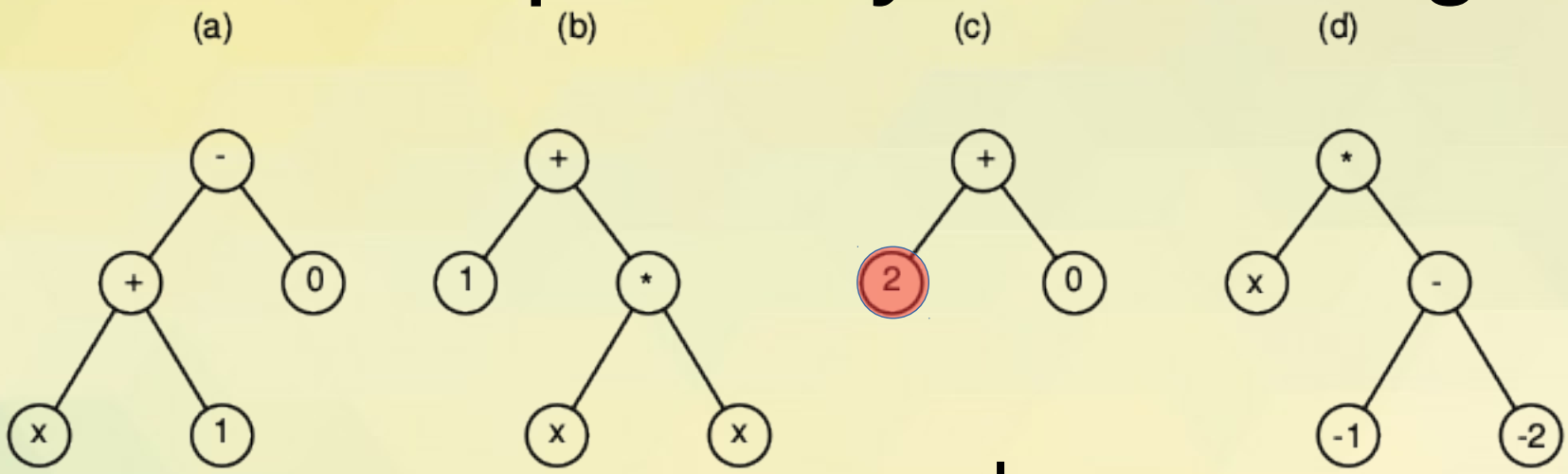
(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a



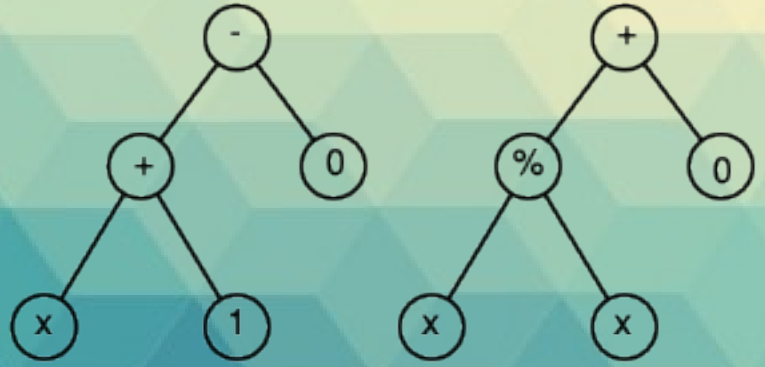
x+1

An Example – Symbolic Regression



(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

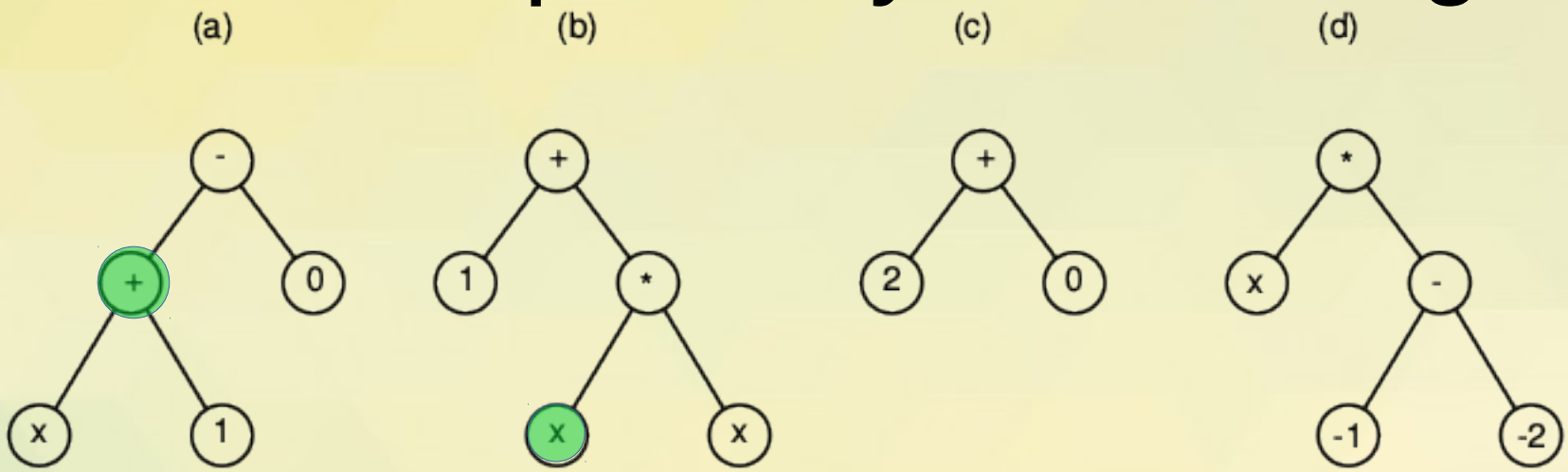
Reproduction: a
Mutation: c



x+1

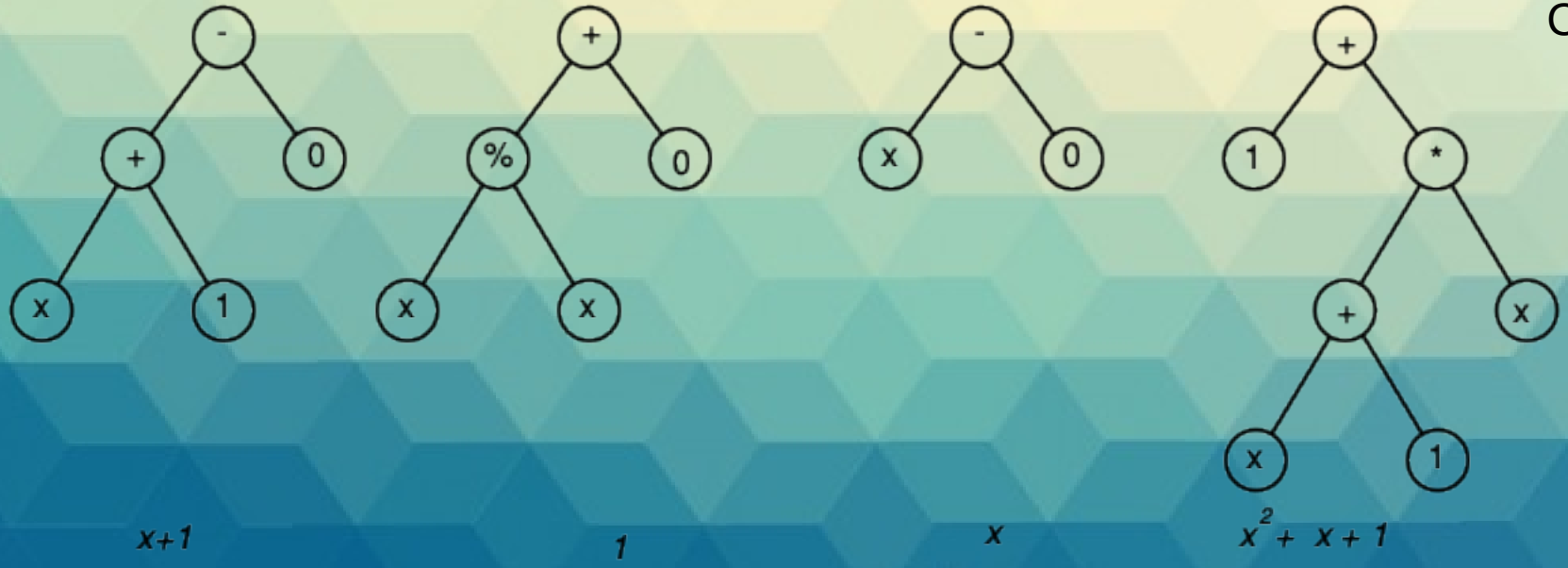
1

An Example – Symbolic Regression



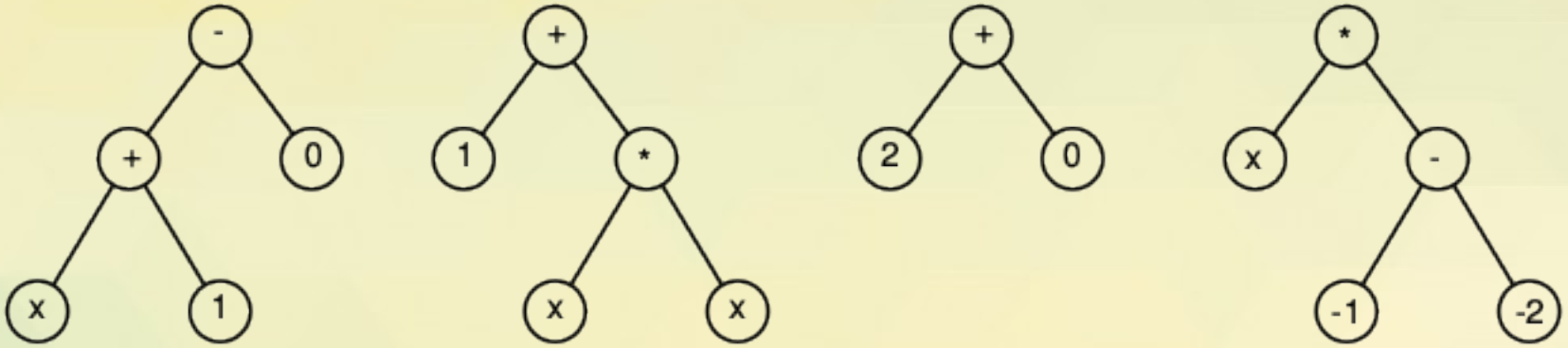
(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a
Mutation: c
Crossover: a, b



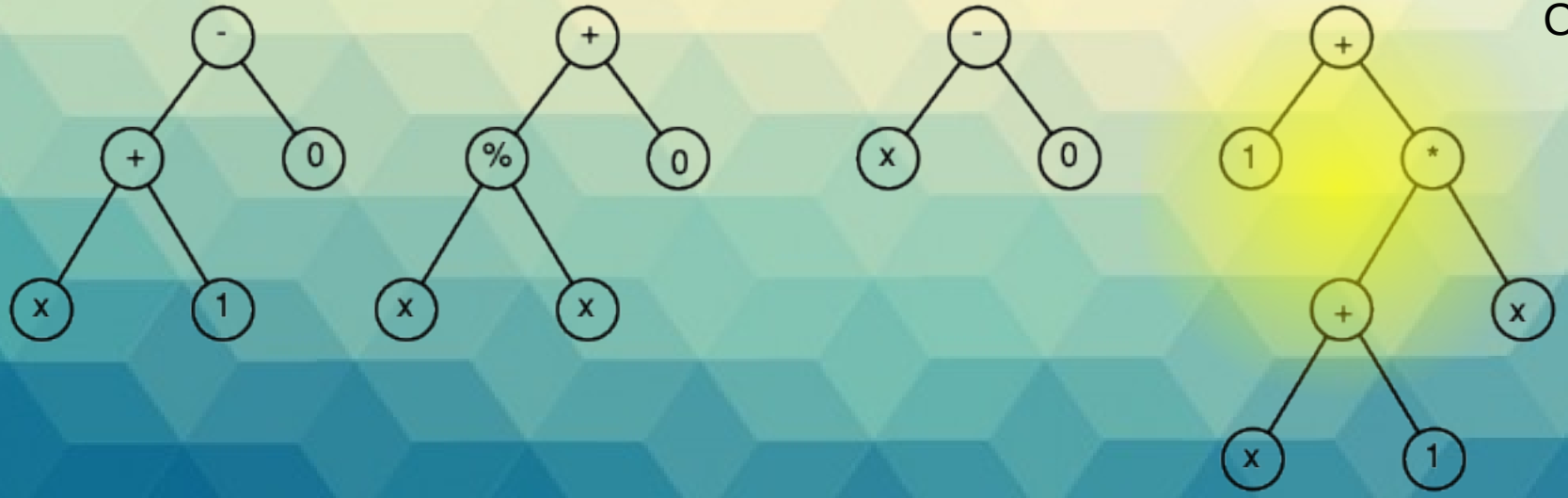
An Example – Symbolic Regression

(a) (b) (c) (d)

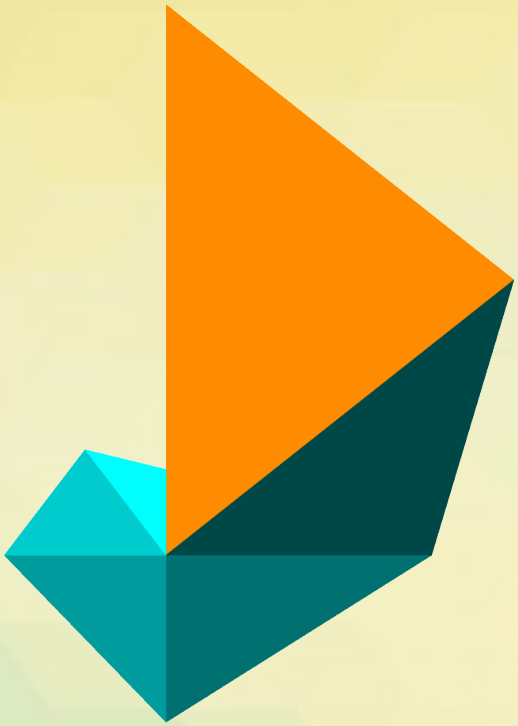


(a) 7.7
(b) 11.0
(c) 17.98
(d) 28.7

Reproduction: a
Mutation: c
Crossover: a, b



x+1 1 x $x^2 + x + 1$



DEAP

DISTRIBUTED EVOLUTIONARY ALGORITHMS IN PYTHON

http://deap.readthedocs.io/
https://github.com/DEAP/deap

DEAP: Evolving 1's

```python
import array
import random
from deap import algorithms
from deap import base
from deap import creator
from deap import tools

creator.create("FitnessMax", base.Fitness, weights=(1.0,))
creator.create("Individual", array.array, typecode='b', fitness=creator.FitnessMax)

toolbox = base.Toolbox()
toolbox.register("attr_bool", random.randint, 0, 1)
toolbox.register("individual", tools.initRepeat, creator.Individual, toolbox.attr_bool, 100)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("evaluate", lambda x:[sum(x)])
toolbox.register("mate", tools.cxTwoPoint)
toolbox.register("mutate", tools.mutFlipBit, indpb=0.05)
toolbox.register("select", tools.selTournament, tournsize=3)

pop = toolbox.population(n=300)
hof = tools.HallOfFame(1)
pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=40,
                               halloffame=hof, verbose=True)
print hof[0]
```

DEAP: Symbolic Regression

```python
import operator
import math
import random

from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from deap import gp

pset = gp.PrimitiveSet("MAIN", 1)
pset.addPrimitive(operator.add, 2)
pset.addPrimitive(operator.sub, 2)
pset.addPrimitive(operator.mul, 2)
pset.addPrimitive(operator.neg, 1)
pset.addEphemeralConstant("rand101", lambda: random.randint(-1,1))
pset.renameArguments(ARG0='x')

creator.create("FitnessMin", base.Fitness, weights=(-1.0,))
creator.create("Individual", gp.PrimitiveTree, fitness=creator.FitnessMin)

toolbox = base.Toolbox()
toolbox.register("expr", gp.genHalfAndHalf, pset=pset, min_=1, max_=2)
toolbox.register("individual", tools.initIterate, creator.Individual, toolbox.expr)
toolbox.register("population", tools.initRepeat, list, toolbox.individual)
toolbox.register("compile", gp.compile, pset=pset)

def evalSymbReg(individual, points):
    func = toolbox.compile(expr=individual)
    sqerrors = ((func(x) - x**4 - x**3 - x**2 - x)**2 for x in points)
    return math.fsum(sqerrors) / len(points),

toolbox.register("evaluate", evalSymbReg, points=[x/10. for x in range(-10,10)])
toolbox.register("select", tools.selTournament, tournsize=3)
toolbox.register("mate", gp.cxOnePoint)
toolbox.register("expr_mut", gp.genFull, min_=0, max_=2)
toolbox.register("mutate", gp.mutUniform, expr=toolbox.expr_mut, pset=pset)

pop = toolbox.population(n=300)
hof = tools.HallOfFame(1)
pop, log = algorithms.eaSimple(pop, toolbox, 0.5, 0.1, 40, halloffame=hof, verbose=True)
print hof[0]
```

DEAP: Symbolic Regression

```
def compile(expr, pset):
    args = ",".join(arg for arg in pset.arguments)
    code = "lambda {args}: {code}".format(args=args, code=str(expr))
    return eval(code, pset.context, {})
```

Python Bytecode (!)

```
function(code, globals[, name[, argdefs[, closure]]])

Create a function object from a code object and a dictionary.
The optional name string overrides the name from the code object.
The optional argdefs tuple specifies the default argument values.
The optional closure tuple supplies the bindings for free variables.
```

```
code(argcount, nlocals, stacksize, flags, codestring, constants, names,
      varnames, filename, name, firstlineno, lnotab[, freevars[, cellvars]])

Create a code object.  Not for the faint of heart.
```

Relevant:

codestring
argcount
nlocals
stacksize
constants
varnames
flags

Less Relevant

freevars
cellvars

Unlikely Relevant

names
kwonlyargcount

Inconsequential

filename
name
firstlineno
lnotab

Flags (!)

CO_OPTIMIZED	**0x0001**	**Was once an important flag**
CO_NEWLOCALS	**0x0002**	new 'locals' dict for execution otherwise uses globals if local object unsupplied
CO_VARARGS	0x0004	For Variable args
CO_VARKEYWORDS	0x0008	For Variable kwargs
CO_NESTED	0x0010	unused
CO_NOFREE	0x0040	unused
CO_GENERATOR	0x0020	For Generators
CO_COROUTINE	0x0080	For Coroutines
CO_ITERABLE_COROUTINE	0x0100	For Coroutines
CO_GENERATOR_ALLOWED	0x1000	unused
CO_FUTURE_DIVISION	0x2000	Moot in Python3.5
CO_FUTURE_ABSOLUTE_IMPORT	0x4000	Moot in Python3.5
CO_FUTURE_WITH_STATEMENT	0x8000	Moot in Python3.5
CO_FUTURE_PRINT_FUNCTION	0x10000	Moot in Python3.5
CO_FUTURE_UNICODE_LITERALS	0x20000	Moot in Python3.5
CO_FUTURE_BARRY_AS_BDFL	0x40000	Easter Egg ¯_(ツ)_/¯
CO_FUTURE_GENERATOR_STOP	0x80000	generator raising StopIteration

Easter Egg

```
>>> from __future__ import braces
  File "<stdin>", line 1
SyntaxError: not a chance
```

```
>>> from __future__ import barry_as_FLUFL
>>> #Friendly Language Uncle For Life (FLUFL)
... #(https://www.python.org/dev/peps/pep-0401/)
... 1 <> 2
True
>>> 1 != 2
  File "<stdin>", line 1
    1 != 2
       ^
SyntaxError: invalid syntax
```

Example Bytecode Decomposition

```
1 from dis import dis, show_code
2
3 def a(x,y,z):
4     return x+y*z-4
5
6 print("\nBYTECODE:")
7 print([z for z in a.__code__.co_code])
8 print("\nDISSASEMBLY:")
9 dis(a)
10 print("\nCODE OBJECT:")
11 show_code(a)
12
```

Documentation for opcodes in dis module documentation

See also opcodes.py

```
BYTECODE:
[124, 0, 0, 124, 1, 0, 124, 2, 0, 20, 23, 100, 1, 0, 24, 83]

DISSASEMBLY:
  4           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                1 (y)
              6 LOAD_FAST                2 (z)
              9 BINARY_MULTIPLY
             10 BINARY_ADD
             11 LOAD_CONST               1 (4)
             14 BINARY_SUBTRACT
             15 RETURN_VALUE

CODE OBJECT:
Name:              a
Filename:          1.py
Argument count:    3
Kw-only arguments: 0
Number of locals:  3
Stack size:        3
Flags:             OPTIMIZED, NEWLOCALS, NOFREE
Constants:
   0: None
   1: 4
Variable names:
   0: x
   1: y
   2: z
```

Dissasembly is elaboration of a.__code__.XYZ

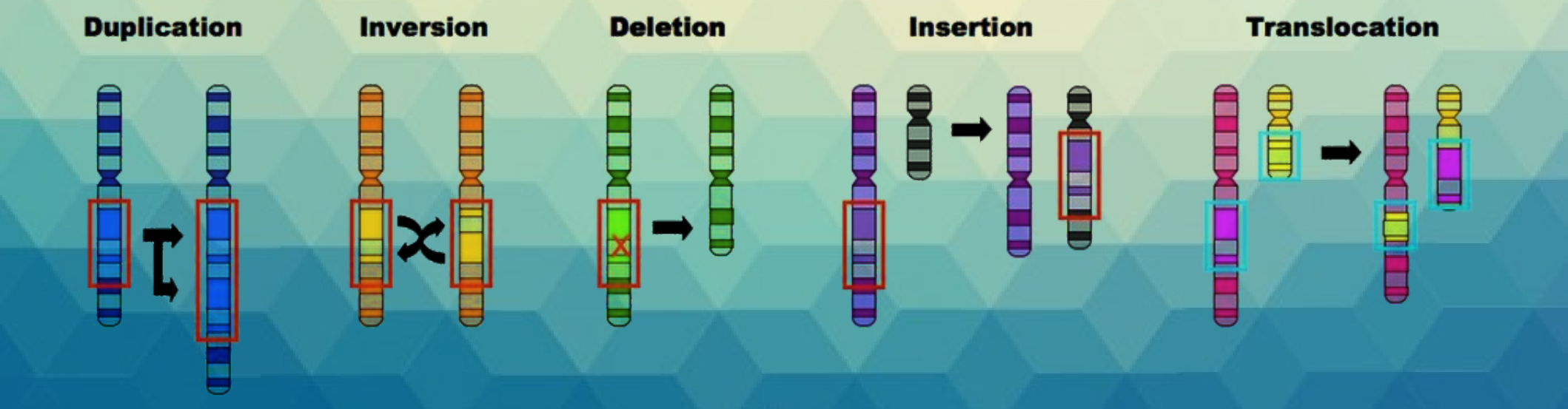
Example Bytecode Reconstitution

```
BYTECODE:
[124, 0, 0, 124, 1, 0, 124, 2, 0, 20, 23, 100, 1, 0, 24, 83]

DISSASEMBLY:
  4           0 LOAD_FAST                0 (x)
              3 LOAD_FAST                1 (y)
              6 LOAD_FAST                2 (z)
              9 BINARY_MULTIPLY
             10 BINARY_ADD
             11 LOAD_CONST               1 (4)
             14 BINARY_SUBTRACT
             15 RETURN_VALUE

CODE OBJECT:
Name:              a
Filename:          1.py
Argument count:    3
Kw-only arguments: 0
Number of locals:  3
Stack size:        3
Flags:             OPTIMIZED
Constants:
   0: None
   1: 4
Variable names:
   0: x
   1: y
   2: z
```

```python
def a(x,y,z):
    return x+y*z-4

from types import CodeType, FunctionType

#code(argcount, kwonlyargcount, nlocals, stacksize, flags, codestring,
#      constants, names, varnames, filename, name, firstlineno,
#      lnotab[, freevars[, cellvars]])
c = CodeType(3,0,3,3,0,
    a.__code__.co_code,
    (None,4),(),("x","y","z"),
    u'bilbo',u'baggins',0,b'')
f = FunctionType(c, {})

```

Linear GP & python – why?

1. is fun/flexible for experimentation
2. is a reasonably well developed VM
3. has potential for distributed GP
4. faster than Tree GP
5. for kicks



Crash python, you must not.

- Stack Underflow causes seg-fault
- Division by zero / other runtime exceptions
- Strange / Random referencing
- Infinite Loops

Remedies

- Program Repairing
- Using safe operations
- Semantic preserving mutation/crossover/generation

Symbolic Regression done with Linear Genetic Programming via Python Bytecode

```python
import random
from deap import algorithms
from deap import base
from deap import creator
from deap import tools
from opcode import opmap
from types import CodeType, FunctionType
from dis import dis
from dis import show_code

operations = (
    (1,(opmap["LOAD_CONST"],0,0)),
    (1,(opmap["LOAD_CONST"],1,0)),
    (1,(opmap["LOAD_CONST"],2,0)),
    (0,(opmap["UNARY_INVERT"],)),
    (-1,(opmap["BINARY_MULTIPLY"],)),
    (-1,(opmap["BINARY_ADD"],)),
    (-1,(opmap["BINARY_SUBTRACT"],)),
    (-1,(opmap["BINARY_AND"],)),
    (-1,(opmap["BINARY_OR"],)),
    (-1,(opmap["BINARY_XOR"],)),
    (1,(opmap["LOAD_FAST"],0,0)),
)
terminate = (opmap["RETURN_VALUE"],)
negones = [o for o in operations if o[0]==-1]
zeros = [o for o in operations if o[0]==0]
ones = [o for o in operations if o[0]==1]

max_len = 40

def fix(individual):
    if not individual.fixed:
        i = 0
        d = 0
        for i in range(max_len,len(individual)):
            individual.pop()
        l = len(individual)
        while i < l:
            if (d+individual[i][0] < 1):
                individual.insert(i,random.choice(ones))
                l = l + 1
            else:
                d = d + individual[i][0]
                i = i + 1
        for i in range(1,d):
            individual.append(random.choice(negones))
        individual.ephemeral = random.randint(0,999999)
        individual.fixed = True
```

```
50 def compile_individual(individual):
51     codebytes = []
52     for d,op_code in individual:
53         codebytes.append(op_code)
54     codebytes.append(terminate)
55     codebytes = [i for sub in codebytes for i in sub]
56     codebytes = bytes(codebytes)
57 
58     #code(argcount, kwonlyargcount, nlocals, stacksize, flags, codestring,
59     #      constants, names, varnames, filename, name, firstlineno,
60     #      lnotab[, freevars[, cellvars]])
61     co_obj = CodeType(1,0,1,max_len,0,codebytes,(1,2,individual.ephemeral),(),("a",),u'file_name',u'code_name',0,b'')
62     B = FunctionType(co_obj, {})
63     return B
64 
65 def evaluate(individual):
66     fix(individual)
67     c = compile_individual(individual)
68     return sum([-(c(i)-i*i+i-4)**2 for i in range(0,10)]),
69 
70 def crossover(i1,i2):
71     i1.fixed = False
72     i2.fixed = False
73     return tools.cxTwoPoint(i1,i2)
74 
75 def mutate(individual, indpb):
76     if random.random() < indpb:
77         individual.fixed = False
78         c = random.randint(0,2)
79         if c==0:
80             return tools.mutShuffleIndexes(individual,1.1)
81         elif c==1:
82             a = random.randint(0,len(individual)-1)
83             b = random.randint(0,len(individual)-1)
84             if a > b:
85                 a,b = b,a
86             del(individual[a:b])
87             return individual,
88         elif c==2:
89             a = random.randint(0,len(individual)-1)
90             del(individual[a:])
91             i = toolbox.individual()
92             [individual.append(a) for a in i]
93             return individual,
94     return individual,
95 
```

```
95
96 creator.create("FitnessMax", base.Fitness, weights=(1.0,))
97 creator.create("Individual", type([]), fitness=creator.FitnessMax, fixed=False, ephemeral=0)
98
99 toolbox = base.Toolbox()
100 toolbox.register("random_operation", random.choice, operations)
101 toolbox.register("random_length", random.randint, 3,8)
102 toolbox.register("individual", lambda c,t,n:c(t() for z in range(n())), creator.Individual, toolbox.random_operation, toolbox.random_length)
103
104 toolbox.register("population", tools.initRepeat, list, toolbox.individual)
105 toolbox.register("evaluate", evaluate)
106 toolbox.register("mate", crossover)
107 toolbox.register("mutate", mutate, indpb=0.05)
108 toolbox.register("select", tools.selTournament, tournsize=3)
109
110 pop = toolbox.population(n=300)
111 hof = tools.HallOfFame(1)
112 pop, log = algorithms.eaSimple(pop, toolbox, cxpb=0.5, mutpb=0.2, ngen=80,
113                                halloffame=hof, verbose=True)
114
115 def print_code(individual):
116     B = compile_individual(individual)
117     print(dis(B))
118     print(show_code(B))
119 def print_values(individual):
120     B = compile_individual(individual)
121     print ([B(i) for i in range(0,10)])
122     print ([i*i-i+4 for i in range(0,10)])
123     print ([-(B(i)-i*i+i-4)**2 for i in range(0,10)])
124
125 print_code(hof[0])
126 print_values(hof[0])
127
```

```
 0 LOAD_CONST               0 (1)
 3 LOAD_CONST               1 (2)
 6 LOAD_FAST                0 (a)
 9 LOAD_FAST                0 (a)
12 BINARY_MULTIPLY
13 BINARY_OR
14 LOAD_FAST                0 (a)
17 BINARY_SUBTRACT
18 BINARY_ADD
19 LOAD_CONST               0 (1)
22 BINARY_ADD
23 RETURN_VALUE
```

TA-DA :-)

(((((a*a)|2)-a)+1)+1) = a^2-a+4

THE END