Amusing Algorithms 🤖

@maxhumber (https://twitter.com/maxhumber)

PyBay 2018

github.com/maxhumber/amusing_algorithms
(https://github.com/maxhumber/amusing_algorithms)

Randall Degges @rdegges · 12h

What even is an algorithm, anyway?

26 77

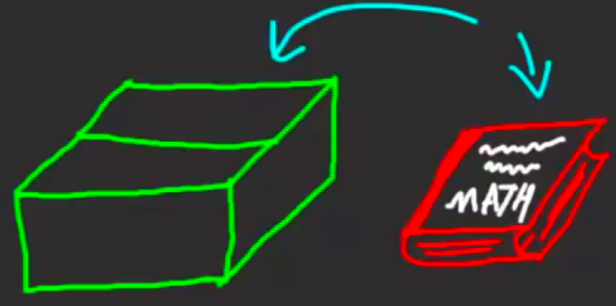
Algorithm

From Wikipedia, the free encyclopedia

For other uses, see Algorithm (disambiguation).

In mathematics and computer science, an **algorithm** (/ˈælgərɪðəm/ *AL-gə-ri-dhəm*) is a self-contained sequence of actions to be performed. Algorithms can perform calculation, data processing and automated reasoning tasks.

What is an Algorithm?



Box Book

Put the book in the box

1) Open the box
2) Pick up the book
3) Put the book inside the box
4) Close the box

Directions

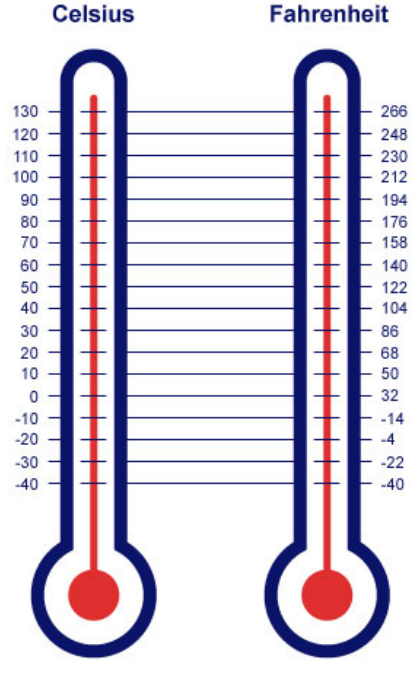
Prep	Cook	Ready In
15 m	35 m	50 m

1. Place sweet potatoes into a large pot and cover with salted water; bring to a boil. Reduce heat to medium-low and simmer until very tender, about 15 minutes. Drain and transfer to a bowl. Mash sweet potatoes and mix in cilantro, chili powder, and salt.

2. Place corn in a microwave-safe bowl and microwave on high until warmed, 1 to 2 minutes.

3. Spread 1/4 cup sweet potato mixture onto 1 tortilla; cover with 1/4 cup black beans and 1 tablespoon corn. Sprinkle 1/4 cup Cheddar cheese atop corn; cover with a tortilla. Repeat with remaining tortillas and fillings.

4. Spray a frying pan with cooking spray and place over medium heat; cook 1 quesadilla in the hot pan until cheese is melted and beans are heated through, 3 to 4 minutes per side. Repeat with remaining quesadillas; slice into quarters.


Celsius
Fahrenheit
130
120
110
100
90
80
70
60
50
40
30
20
10
0
-10
-20
-30
-40
266
248
230
212
194
176
158
140
122
104
86
68
50
32
-14
-4
-22
-40

Machine Learning Algorithms *(sample)*

	Unsupervised	Supervised

Continuous	• Clustering & Dimensionality Reduction
	  ○ SVD
	  ○ PCA
	  ○ K-means
		• Regression
		  ○ Linear
		  ○ Polynomial
		• Decision Trees
		• Random Forests

Categorical	• Association Analysis
	  ○ Apriori
	  ○ FP-Growth
	• Hidden Markov Model
		• Classification
		  ○ KNN
		  ○ Trees
		  ○ Logistic Regression
		  ○ Naive-Bayes
		  ○ SVM

In ‘Amusing Algorithms’ we’ll cut through the math and try to understand the mechanics of a few interesting and useful algorithms. We’ll use **Jupyter** 📒 to expose data structures, intermediate steps, and simulations of various algorithms. And we’ll try to use algorithms to answer real life questions like how to **find love** ❤️ in a crowded bar, how to **buy the best scalpers tickets** 🎟️ at a baseball game, and how to figure out when **you should leave your job** 🤦🏻‍♀️.

- Optimal Stopping 🛑
- Stable Marriage 💒
- Simulated Annealing ⚔️

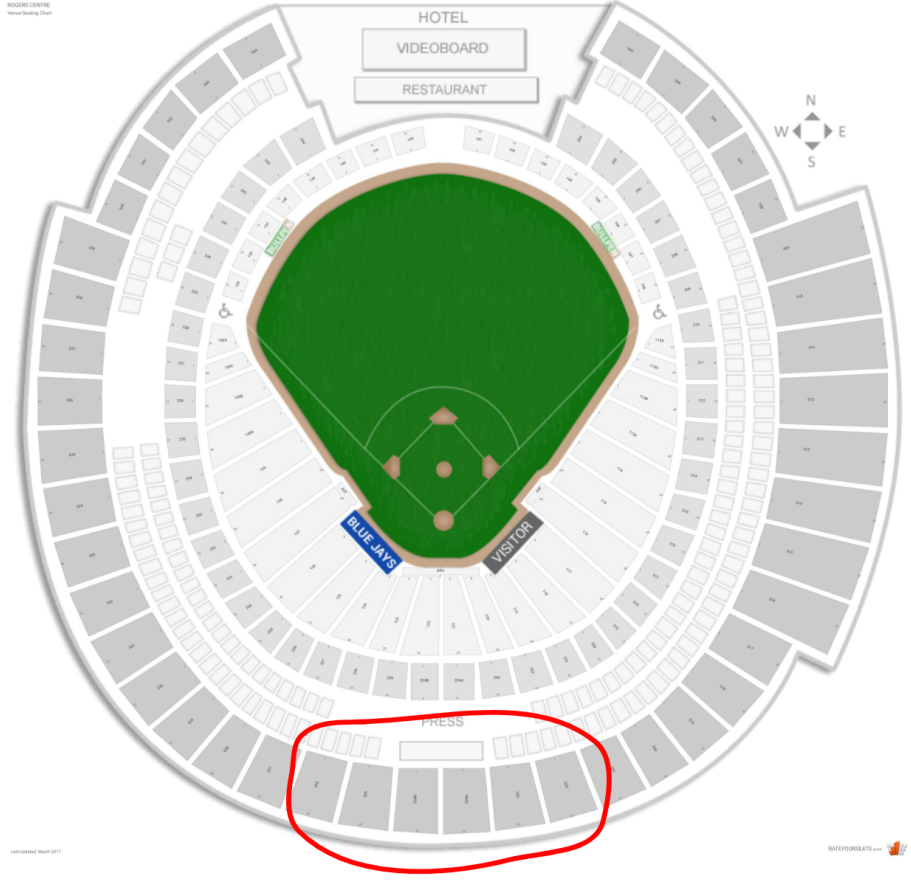
Optimal Stopping 🛑

Imagine an administrator who wants to hire the best secretary out of n rankable applicants for a position. The applicants are interviewed one by one in random order. A decision about each particular applicant is to be made immediately after the interview. Once rejected, an applicant cannot be recalled. During the interview, the administrator can rank the applicant among all applicants interviewed so far, but is unaware of the quality of yet unseen applicants. The question is about the optimal strategy to maximize the probability of selecting the best applicant.

37%


TORONTO
BLUE JAYS
TM


I HAVE TICKETS


ROGERS CENTRE
Venue Seating Chart
HOTEL
VIDEOBOARD
RESTAURANT
N
W
E
S
BLUE JAYS
VISITOR
PRESS
Last Updated: March 2017
RATEYOURSEATS.com

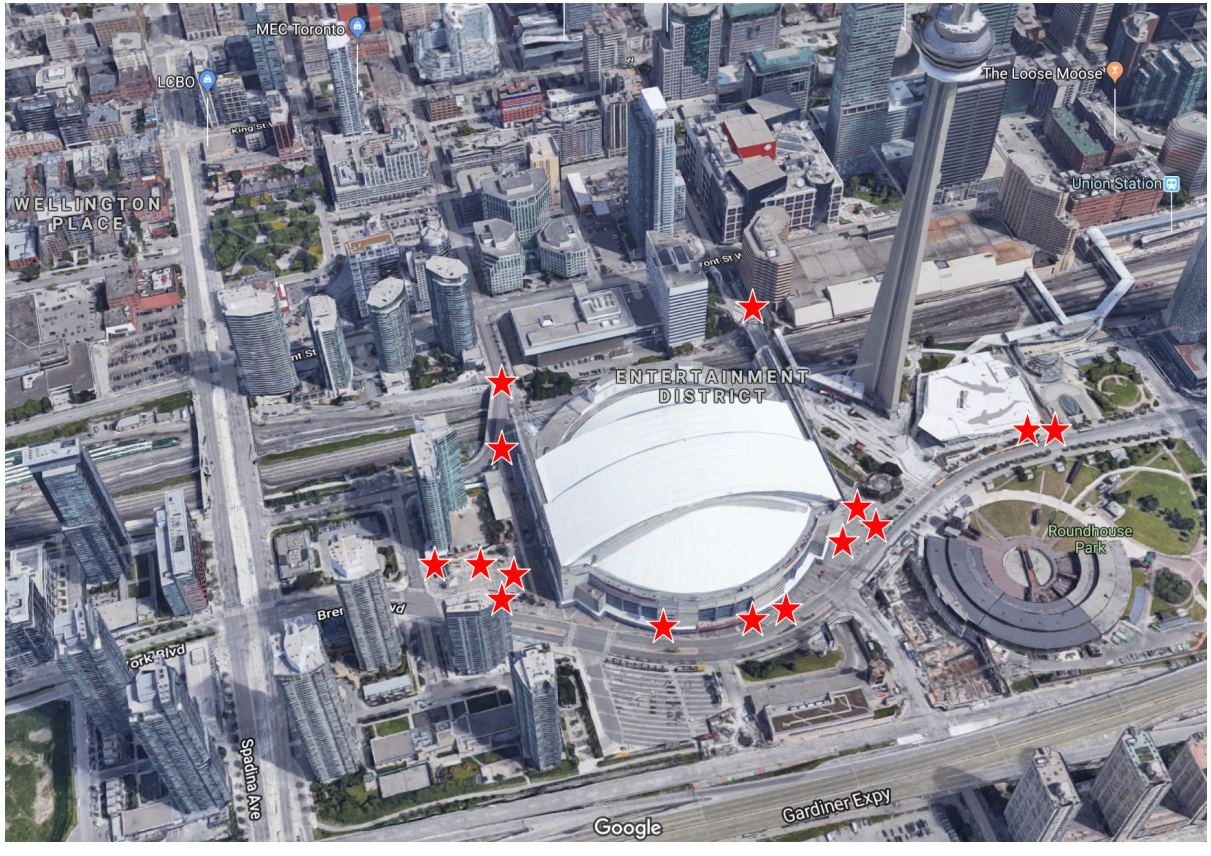
In [1]:
```
low = 8
high = 24
N = 15
```

In [2]:
```
import random
random.seed(14)

options = [random.randint(low, high) for i in range(N)]

print(options)
```

```
[11, 24, 15, 16, 16, 17, 10, 22, 17, 22, 20, 20, 11, 16, 15]
```

In [3]:
```
for _ in range(5):
    options = [random.randint(low, high) for i in range(N)]
    print(options, '\n')
```

```
[18, 19, 16, 19, 24, 12, 13, 16, 13, 8, 10, 11, 18, 8, 10]

[16, 14, 20, 20, 22, 11, 11, 19, 13, 11, 23, 24, 14, 16, 22]

[14, 23, 17, 24, 16, 11, 11, 10, 16, 16, 11, 8, 13, 21, 11]

[24, 10, 21, 23, 13, 20, 22, 17, 23, 22, 21, 21, 11, 16, 23]

[20, 15, 22, 23, 10, 12, 23, 16, 22, 8, 20, 21, 8, 19, 19]
```



In [4]:
```
random.seed(14)
options = [random.randint(low, high) for i in range(N)]
print(options)
```

```
[11, 24, 15, 16, 16, 17, 10, 22, 17, 22, 20, 20, 11, 16, 15]
```

1. Estimate how many possible options there are
2. Multiply that # by `1/e` or ~ 37%
3. Evaluate that # of options to establish a benchmark
4. Continue until you find an option that exceeds the benchmark

In [5]:
```
import math
skip = int(len(options) / math.e)

bench = options[0]
for i, option in enumerate(options):
    if i < skip:
        if option <= bench:
            bench = option
    else:
        if option <= bench:
            break
```

In [6]:
```
print(skip)
```

```
5
```

In [7]:
```
print(f'Options: {options}')
print(f'Benchmark: {bench}')
print(f'Choice: {option} at position {i+1}/{len(options)}')
```

```
Options: [11, 24, 15, 16, 16, 17, 10, 22, 17, 22, 20, 20, 11, 16, 15]
Benchmark: 11
Choice: 10 at position 7/15
```

Refactor 🚨

In [8]:
```
skip = int(len(options) / math.e)
bench = min(options[:skip])

for option in options[skip:]:
    if option <= bench:
        break
```

In [9]:
```
def create_options(low=8, high=24, N=15):
    return [random.randint(low, high) for i in range(N)]
```

In [10]:
```
print(create_options(8, 24, 15))
print('\n')
print(create_options(60, 100, 10))
```

```
[18, 19, 16, 19, 24, 12, 13, 16, 13, 8, 10, 11, 18, 8, 10]


[77, 73, 84, 85, 97, 88, 98, 66, 67, 97]
```

Refactor 🚨

In [11]:
```
def buy(options, strategy='optimal'):
    if strategy == 'first':
        choice = options[0]
    if strategy == 'random':
        choice = random.choice(options)
    if strategy == 'optimal':
        skip = int(len(options) / math.e)
        bench = min(options[:skip])
        for option in options[skip:]:
            if option <= bench:
                break
        choice = option
    return 1 if choice == min(options) else 0
```

Simulate ⏳

In [12]:
```
outcomes = []
for i in range(1000):
    outcome = buy(create_options(low=8, high=24, N=15), 'optimal')
    outcomes.append(outcome)
sum(outcomes) / len(outcomes)
```

Out[12]: 0.488

In [13]:
```
outcomes = []
for i in range(1000):
    outcome = buy(create_options(8, 24, 15), 'first')
    outcomes.append(outcome)
sum(outcomes) / len(outcomes)
```

Out[13]: 0.088

In [14]:
```
outcomes = []
for i in range(1000):
    outcome = buy(create_options(8, 24, 15), 'random')
    outcomes.append(outcome)
sum(outcomes) / len(outcomes)
```

Out[14]: 0.088

```
skip = 37%

for option in options[skip:]:
    if option <= bench:
        break
```

Stable Marriage 

Given n men and n women, where each person has ranked all members of the opposite sex in order of preference, marry the men and women together such that there are no two people of opposite sex who would both rather have each other than their current partners. When there are no such pairs of people, the set of marriages is deemed stable.

Friends 👫👫

Data Structures 👷‍♂️

In [15]:
```
class Person:
    def __init__(self, name, preferences):
        self.name = name
        self.partner = None
        self.preferences = preferences

    def __repr__(self):
        if self.partner:
            return f'{self.name} ⚮ {self.partner}'
        else:
            return f'{self.name} ⌀'
```

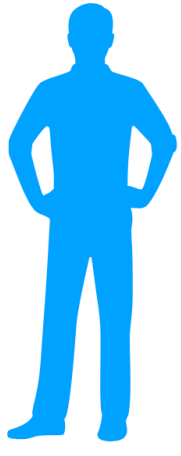
In [16]:
```
ross = Person('Ross', ['Rachel', 'Phoebe', 'Monia'])
print('Name:', ross)
print('Partner:', ross.partner)
```

```
Name: Ross ⌀
Partner: None
```

In [17]:
```
ross.partner = 'Rachel'
print(ross)
```

```
Ross ⚮ Rachel
```

Functions 



ask()

accept()

In [18]:
```
import copy

class Alpha(Person):
    def __init__(self, name, preferences):
        super().__init__(name, preferences)
        self.not_asked = copy.deepcopy(preferences)

    def ask(self):
        return self.not_asked.pop(0)
```

In [19]:
```
class Beta(Person):
    def __init__(self, name, preferences):
        super().__init__(name, preferences)

    def accept(self, suitor):
        return self.partner is None or (
            self.preferences.index(suitor) <
            self.preferences.index(self.partner)
        )
```

Preferences 💏

In [20]:
```
people = {
    "Men": {
        "Ross": ["Rachel", "Phoebe", "Monica"],
        "Chandler": ["Rachel", "Monica", "Phoebe"],
        "Joey": ["Phoebe", "Rachel", "Monica"]
    },
    "Women": {
        "Rachel": ["Joey", "Ross", "Chandler"],
        "Phoebe": ["Ross", "Chandler", "Joey"],
        "Monica": ["Joey", "Chandler", "Ross"]
    }
}
```

Initialize 🌱

In [21]:
```
def setup():
    global alphas
    global betas

    alphas = {}
    for key, value in people.get('Men').items():
        alphas[key] = Alpha(key, value)
    betas = {}
    for key, value in people.get('Women').items():
        betas[key] = Beta(key, value)
```

In [22]:
```
setup()
print('A:', alphas)
print('B:', betas)
```

```
A: {'Ross': Ross ⌀, 'Chandler': Chandler ⌀, 'Joey': Joey ⌀}
B: {'Rachel': Rachel ⌀, 'Phoebe': Phoebe ⌀, 'Monica': Monica ⌀}
```

In [23]:
```
unmatched = list(alphas.keys())
print(unmatched)
```

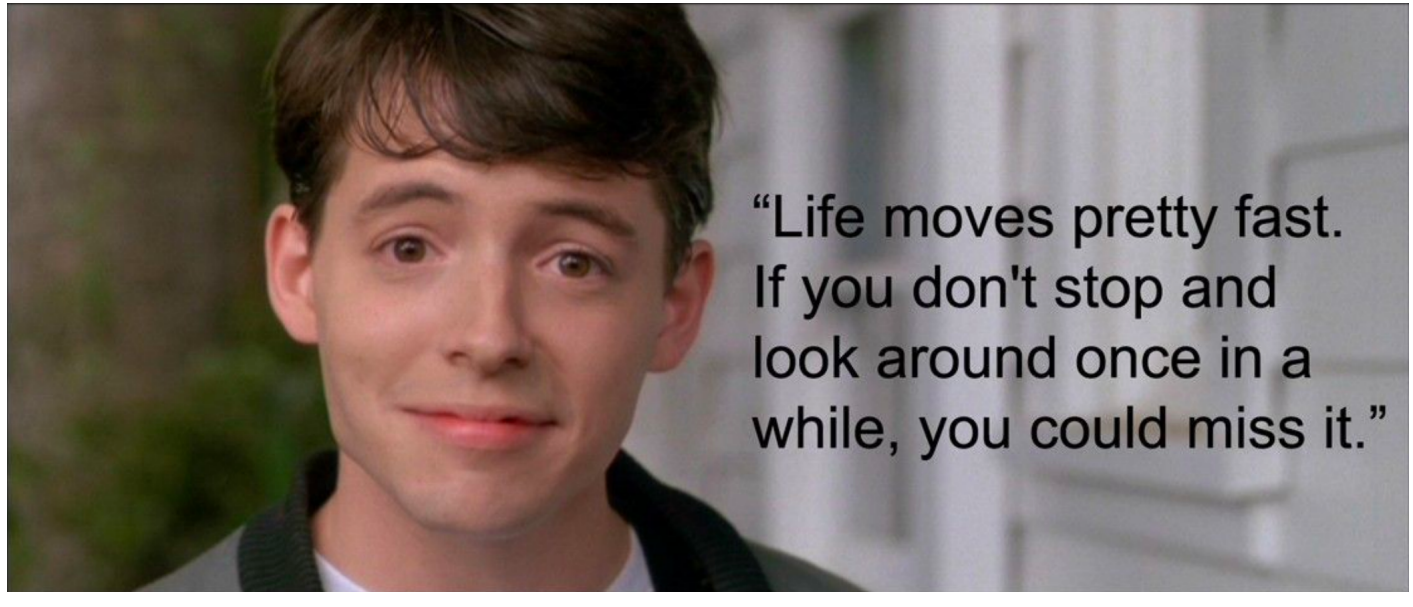
```
['Ross', 'Chandler', 'Joey']
```

1. Each unengaged man proposes to the woman he prefers most
2. Each woman says "maybe" to the suitor she most prefers and "no" to all other suitors
3. Continue while there are still unengaged men

In [24]:
```
while unmatched:
    alpha = alphas[random.choice(unmatched)]
    beta = betas[alpha.ask()]
    if beta.accept(alpha.name):
        if beta.partner:
            ex = alphas[beta.partner]
            ex.partner = None
            unmatched.append(ex.name)
        unmatched.remove(alpha.name)
        alpha.partner, beta.partner = beta.name, alpha.name
```

In [25]:
```
print(alphas)
print(betas)
```

```
{'Ross': Ross ⚮ Rachel, 'Chandler': Chandler ⚮ Monica, 'Joey': Joey ⚮ Phoebe}
{'Rachel': Rachel ⚮ Ross, 'Phoebe': Phoebe ⚮ Joey, 'Monica': Monica ⚮ Chandle
r}
```


“Life moves pretty fast.
If you don't stop and
look around once in a
while, you could miss it.”

In [26]:
```
while unmatched:
    alpha = alphas[random.choice(unmatched)]
    beta = betas[alpha.ask()]
    if beta.accept(alpha.name):
        if beta.partner:
            ex = alphas[beta.partner]
            ex.partner = None
            unmatched.append(ex.name)
        unmatched.remove(alpha.name)
        alpha.partner, beta.partner = beta.name, alpha.name
```

In [27]:

```
import time
random.seed(1993)
setup()
unmatched = list(alphas.keys())

while unmatched:
    alpha = alphas[random.choice(unmatched)]
    beta = betas[alpha.ask()]
    print(f'{alpha.name} asks {beta.name}')
    time.sleep(0.8)
    if beta.accept(alpha.name):
        print(f'{beta.name} accepts')
        if beta.partner:
            ex = alphas[beta.partner]
            print(f'{beta.name} dumps {ex.name}')
            ex.partner = None
            unmatched.append(ex.name)
        unmatched.remove(alpha.name)
        alpha.partner, beta.partner = beta.name, alpha.name
    else:
        print(f'{beta.name} rejects')
    time.sleep(0.8)
    print()
print('Everyone is matched! And things are stable 💪')
```

```
Chandler asks Rachel
Rachel accepts

Joey asks Phoebe
Phoebe accepts

Ross asks Rachel
Rachel accepts
Rachel dumps Chandler

Chandler asks Monica
```

In [28]:
```
def print_pairings(people):
    for p in people.values():
        if p.partner:
            print(f'{p.name} is paired with {p.partner} ({p.preferences.index(p.pa
rtner) + 1})')
        else:
            print(f'{p.name} is not paired')
```

In [29]:
```
print_pairings(alphas)
print('\n')
print_pairings(betas)
```

```
Ross is paired with Rachel (1)
Chandler is paired with Monica (2)
Joey is paired with Phoebe (1)


Rachel is paired with Ross (2)
Phoebe is paired with Joey (3)
Monica is paired with Chandler (2)
```

In [30]:
```
alphas = {}
for key, value in people.get('Women').items():
    alphas[key] = Alpha(key, value)
betas = {}
for key, value in people.get('Men').items():
    betas[key] = Beta(key, value)

unmatched = list(alphas.keys())
```

In [31]:
```
while unmatched:
    alpha = alphas[random.choice(unmatched)]
    beta = betas[alpha.ask()]
    if beta.accept(alpha.name):
        if beta.partner:
            ex = alphas[beta.partner]
            ex.partner = None
            unmatched.append(ex.name)
        unmatched.remove(alpha.name)
        alpha.partner, beta.partner = beta.name, alpha.name
```

In [32]:
```
print_pairings(alphas)
print('\n')
print_pairings(betas)
```

```
Rachel is paired with Joey (1)
Phoebe is paired with Ross (1)
Monica is paired with Chandler (2)


Ross is paired with Phoebe (2)
Chandler is paired with Monica (2)
Joey is paired with Rachel (2)
```

> *"Pearls don’t lie on the seashore. If you want one, you must dive for it."*

— Chinese proverb

> *"Opportunity dances with those on the dance floor."*

— Anonymous

Simulated Annealing ⚔️

Aims of Annealing

- 1. Increase ductility
- 2. Reduce hardness
- 3. Improving formability
- 4. Recrystallize cold worked (strain hardened) metals
- 5. Remove internal stresses
- 6. Increase toughness
- 7. Decrease brittleness
- 8. Increase machinability
- 9. Decrease electrical resistance
- 10. Improve magnetic properties

Imagine you are situated at the bottom of a mountain, and you want to make your way up by foot to the highest point in the valley. But you cannot see beyond your feet, so your algorithm is simply to keep heading uphill. You do that, and eventually, half way up the mountain, you end up at the top of a small peak, from which all paths lead down. As far as you can nearsightedly tell, you've reached the highest point. But it's a local maximum, not truly the top of the mountain, and you're not where you need to be.

What would be a better algorithm for someone with purely local vision? Simulated annealing, inspired by sword makers.

Setup 💼

In [33]:
```
def clamp(x, low, high):
    return max(min(x, high), low)
```

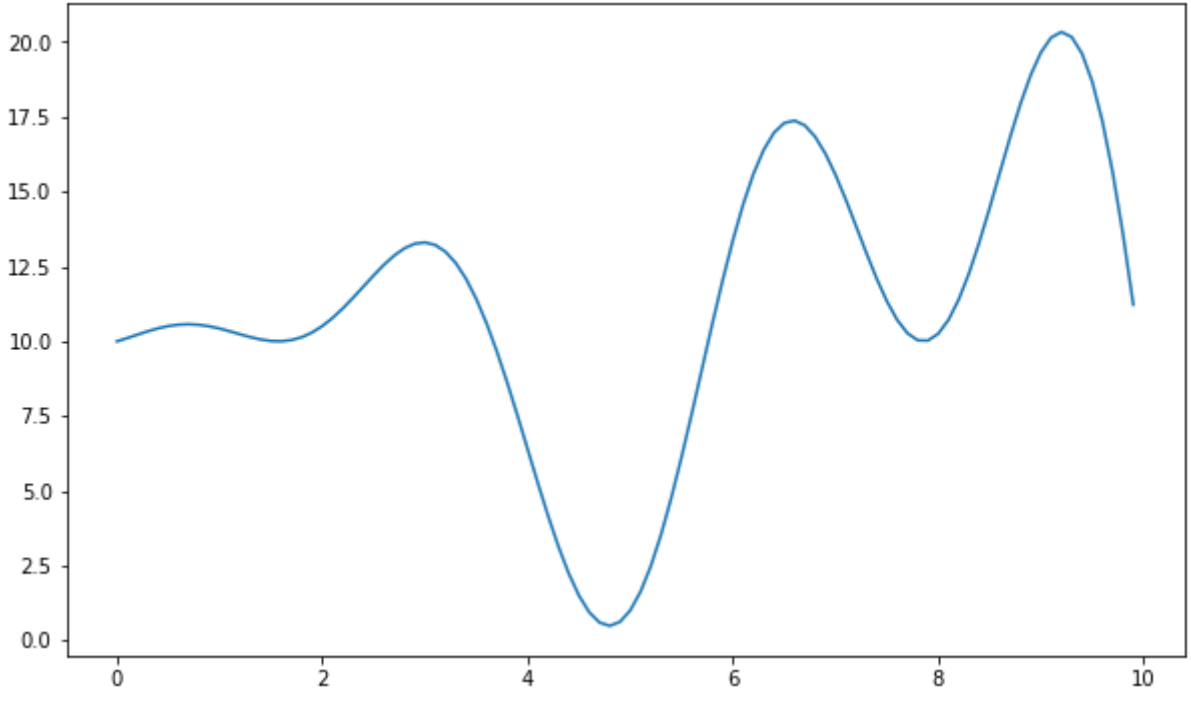
In [34]:
```
def linspace(low, high, steps):
    return [low + x * (high - low)/steps for x in range(steps)]
```

In [35]:
```
import math

def f(x):
    x = clamp(x, 0, 10)
    return x * math.sin(x) + x * math.cos(2 * x) + 10
```

In [36]:
```
xs = linspace(0, 10, 100)
ys = [f(x) for x in xs]
```

In [37]:
```
from matplotlib import pyplot as plt
%matplotlib inline

plt.figure(figsize=(10, 6))
plt.plot(xs, ys)
```

Out[37]:
```
[<matplotlib.lines.Line2D at 0x116129978>]
```



In [38]:
```
x = 4
fx = f(x)
print(fx)
```

```
6.390789883533833
```

In [39]:
```
x = 5
fx = f(x)
print(fx)
```

```
1.0100209813020449
```

In [40]:
```
import random

random.seed(1993)
x2 = x + random.choice([-1, 1]) * 0.001
fx2 = f(x2)
print(fx)
print(fx2)
```

```
1.0100209813020449
1.015093665555547
```

In [41]:
```
if fx2 > fx:
    print('fx2 is better')
    x, fx = x2, fx
else:
    print('fx2 is worse')
```

```
fx2 is better
```

In [42]:
```
def hill_climb(x, f, steps=1000):
    fx = f(x)
    for i in range(steps):
        x2 = x + random.choice([-1, 1]) * 0.01
        fx2 = f(x2)
        if fx2 > fx:
            x, fx = x2, fx2
    return x, fx
```

In [43]:
```
hill_climb(4, f)
```

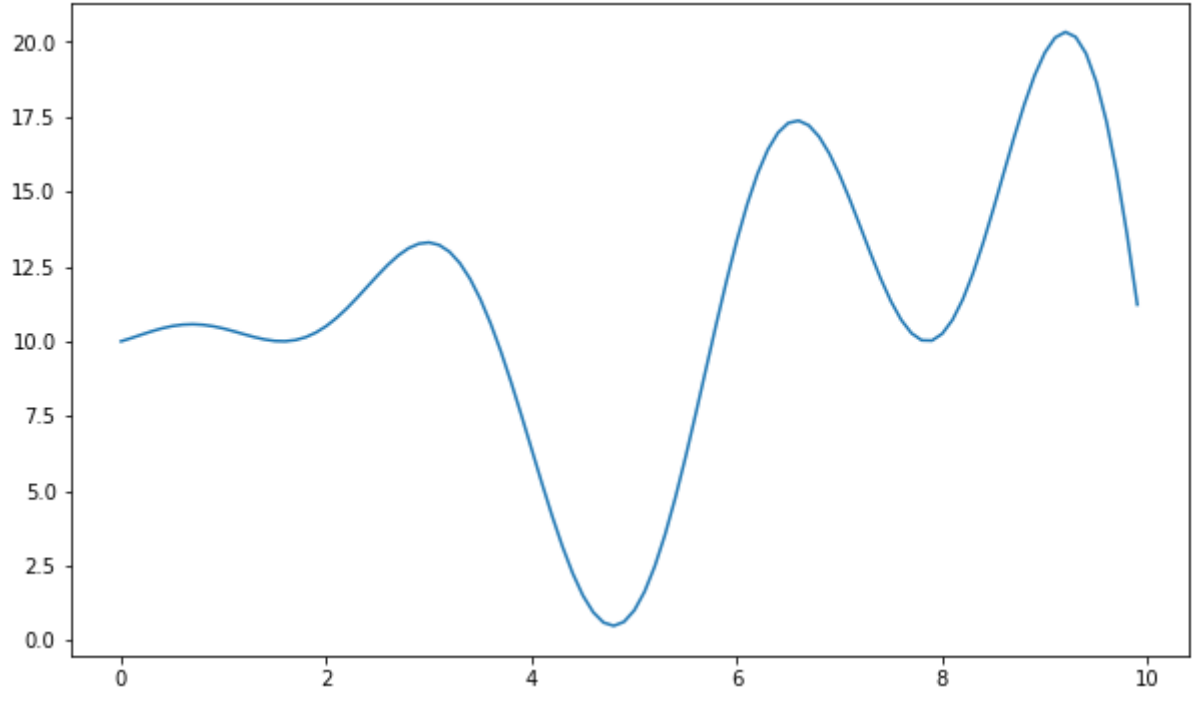
Out[43]: (2.9800000000000217, 13.30517440563907)

In [44]:
```
hill_climb(5, f)
```

Out[44]: (6.579999999999966, 17.378756955031456)

In [45]:
```
plt.figure(figsize=(10, 6))
plt.plot(xs, ys)
# plt.scatter(4, f(4), c='r')
# plt.scatter(5, f(5), c='k')
```

Out[45]:
```
[<matplotlib.lines.Line2D at 0x1162fbe10>]
```




CAREER

When the ascent in the simulated annealing algorithm takes you to some maximum, you sometimes take a chance and shake things up at random, in the hope that by sometimes shaking yourself out of the local maximum and temporarily moving lower, (which is not where you ultimately want to be), you may then find your way to a lower and more stable global maximum.

Pseudocode

```
Let s = s0
For k = 0 through kmax (exclusive):
T ← temperature(k / kmax)
Pick a random neighbour, snew ← neighbour(s)
If P(E(s), E(snew), T) ≥ random(0, 1):
s ← snew
Output: the final state s
```

In [46]:
```
def prob(old, new, temperature):
    # if trying to minimize: math.exp(-(new - old) / temperature)
    return math.exp(-(old - new) / temperature)
```

In [47]:
```
old = 10
new = 9
```

In [48]:
```
print(prob(old, new, temperature=10))
print(prob(old, new, temperature=1))
```

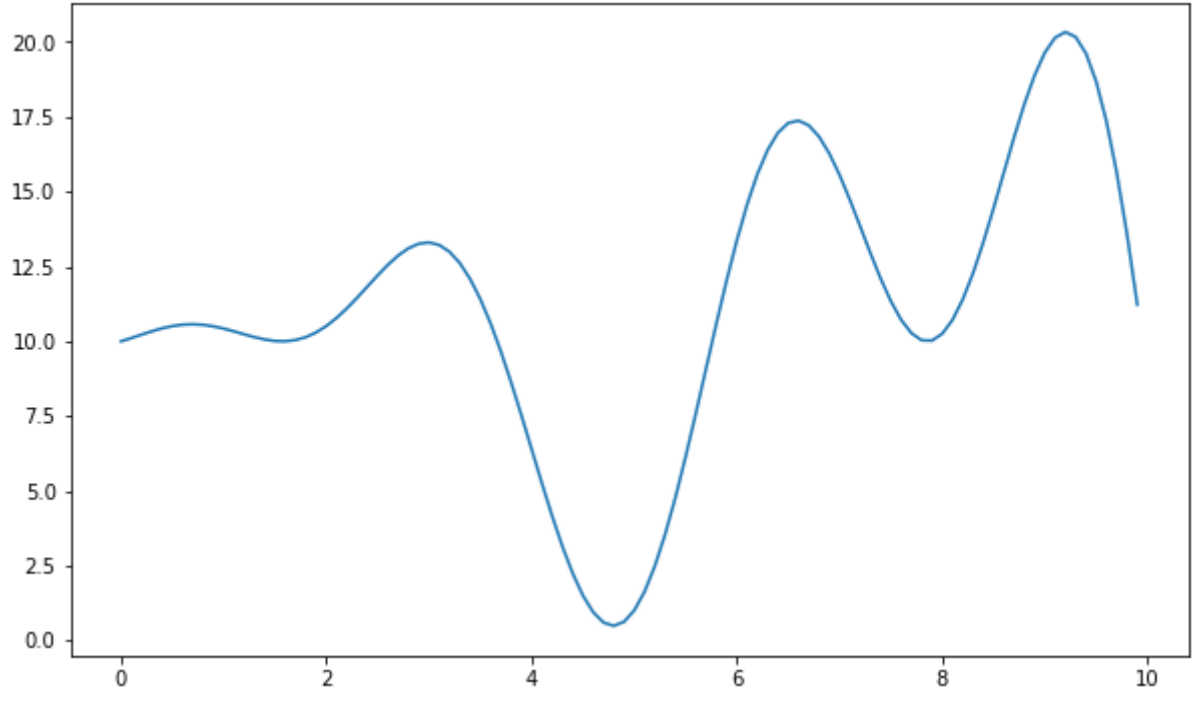
```
0.9048374180359595
0.36787944117144233
```

In [49]:
```
temperature = 1000
a = 0.99
for _ in range(1000):
    print(temperature)
    temperature *= a
```

```
1000
990.0
980.1
970.299
960.59601
950.9900498999999
941.480149401
932.0653479069899
922.74469442792
913.5172474836407
904.3820750088043
895.3382542587163
886.3848717161292
877.5210229989679
868.7458127689781
860.0583546412884
851.4577710948755
842.9431933839268
834.5137614500875
826.1686238355866
817.9069375972307
809.7278682212584
801.6305895390458
793.6142836436553
785.6781408072187
777.8213593991466
770.0431458051551
762.3427143471035
754.7192872036325
747.1720943315961
739.7003733882801
```

In [51]:
```
plt.figure(figsize=(10, 6))
plt.plot(xs, ys)
```

Out[51]:
```
[<matplotlib.lines.Line2D at 0x116841c88>]
```



In [52]:
```
import pathlib
import re
import random
import imageio
```

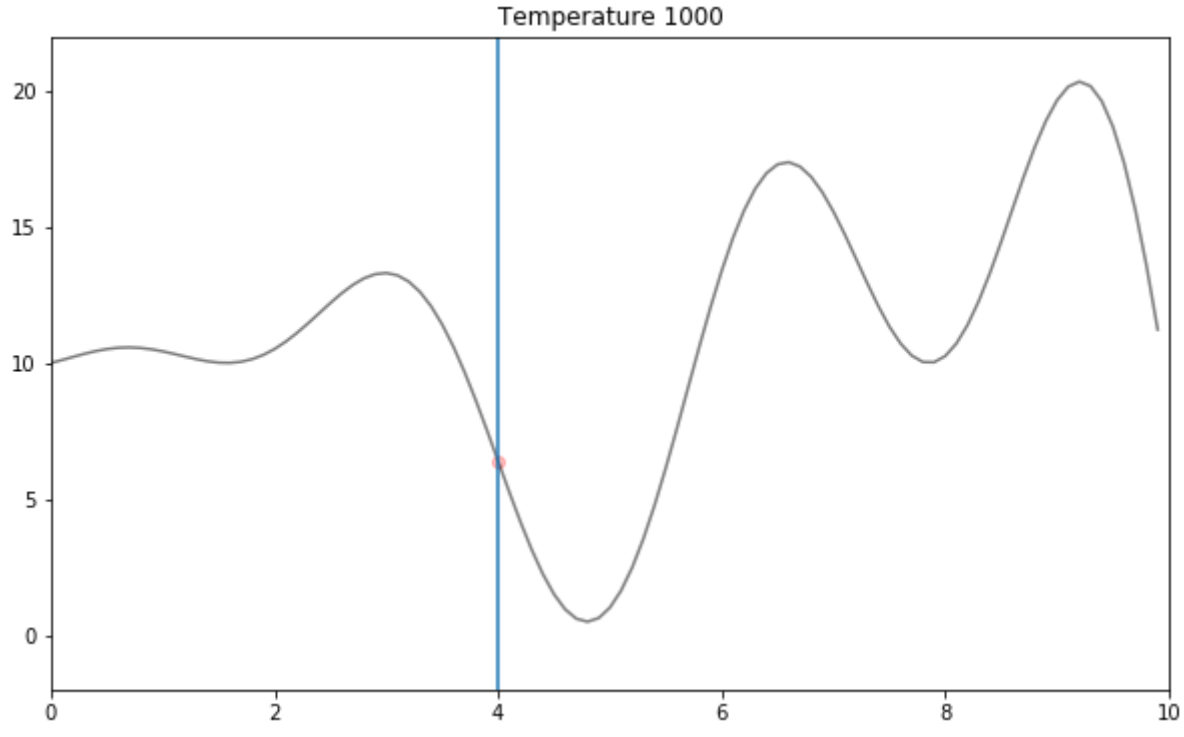
In [53]:
```
random.seed(1993)
current = 4
temperature = 1000
alpha = 0.90

x = [current]
y = [f(current)]
t = [temperature]

for _ in range(99):
    new = random.uniform(0, 10)
    if prob(f(current), f(new), temperature) >= random.random():
        current = new
    x.append(current)
    y.append(f(current))
    t.append(temperature)
    temperature *= alpha
```

In [54]:
```
print(len(x))
print(len(y))
print(len(t))
```

```
100
100
100
```

In [55]:
```
# create temp path for deletion later
p = pathlib.Path('temp/')
p.mkdir(parents=True, exist_ok=True)

# build all of the individual graphs
for i in range(len(x)):
    plt.figure(figsize=(10, 6))
    plt.plot(xs, ys, c='k', alpha=0.5)
    plt.scatter(x[:i], y[:i], c='r', alpha=0.25)
    try:
        plt.axvline(x[:i][-1])
        plt.title(f'Temperature {round(t[:i][-1], 4)}')
    except IndexError:
        pass
    plt.ylim(-2, 22)
    plt.xlim(0, 10)
    plt.savefig(p / f'{i}.png')
```

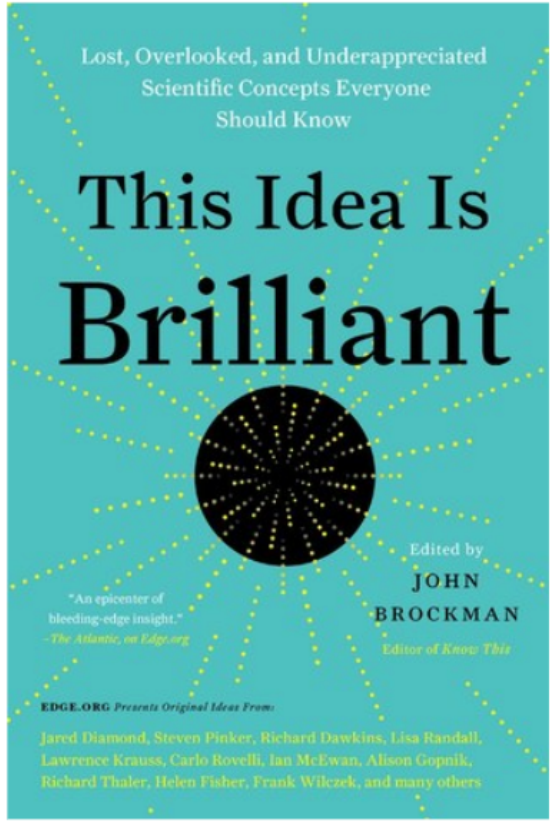
```
/Users/max/anaconda3/lib/python3.6/site-packages/matplotlib/pyplot.py:537: Run
timeWarning: More than 20 figures have been opened. Figures created through th
e pyplot interface (`matplotlib.pyplot.figure`) are retained until explicitly
closed and may consume too much memory. (To control this warning, see the rcPa
ram `figure.max_open_warning`).
  max_open_warning, RuntimeWarning)
```

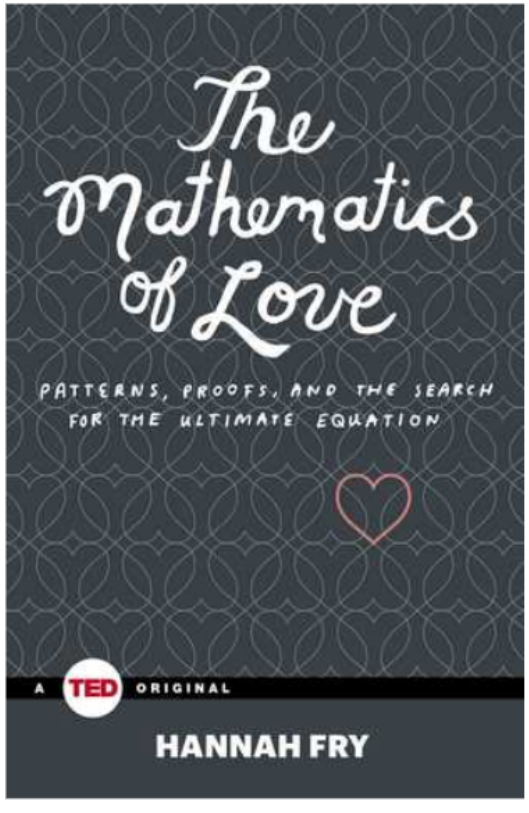

Temperature 1000
20
15
10
5
0
0
2
4
6
8
10

In [56]:
```
filenames = [str(x) for x in p.glob('*.png')]
filenames.sort(key=lambda x: int(re.sub('\D', '', x)))

# build gif and clean up pictures
images = []
for filename in filenames:
    images.append(imageio.imread(filename))
imageio.mimsave('images/sim_anneal.gif', images, duration=0.1)
```

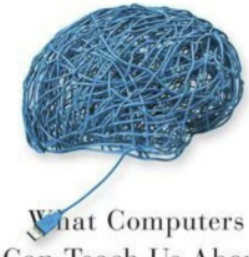
In [57]:
```
[x.unlink() for x in p.iterdir()]
p.rmdir()
```

Simulated annealing employs judicious volatility in the hope that it will be beneficial. In an impossibly complex world, we should perhaps shun temporary stability and instead be willing to tolerate a bit of volatility in order to find a greater stability thereafter.


Lost, Overlooked, and Underappreciated
Scientific Concepts Everyone
Should Know
This Idea Is
Brilliant
Edited by
JOHN
BROCKMAN
Editor of Know This
"An epicenter of
bleeding-edge insight."
–The Atlantic, on Edge.org
EDGE.ORG Presents Original Ideas From:
Jared Diamond, Steven Pinker, Richard Dawkins, Lisa Randall,
Lawrence Krauss, Carlo Rovelli, Ian McEwan, Alison Gopnik,
Richard Thaler, Helen Fisher, Frank Wilczek, and many others


The Mathematics of Love
PATTERNS, PROOFS, AND THE SEARCH FOR THE ULTIMATE EQUATION
A TED ORIGINAL
HANNAH FRY

Algorithms to Live By



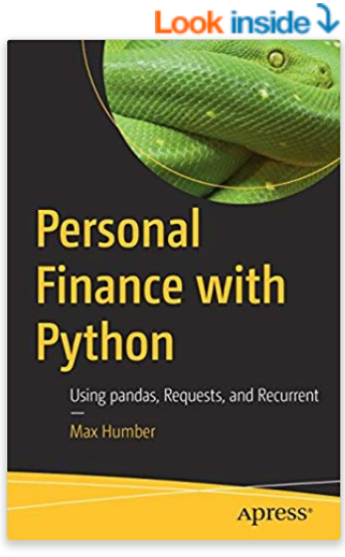
What Computers Can Teach Us About Solving Human Problems

Brian Christian and Tom Griffiths

Shameless Plug

https://www.amazon.com/Personal-Finance-Python-Requests-Recurrent/dp/148423801X/

‹ Back to search results for "max humber"



See all 2 images

Personal Finance with Python: Using pandas, Requests, and Recurrent Paperback – July 21, 2018

by Max Humber (Author)

Be the first to review this item

› See all 2 formats and editions

Kindle	Paperback
$24.56	**$29.99**
Read with Our **Free App**	6 Used from $21.85 18 New from $21.56

Deal with data, build up financial formulas in code from scratch, and evaluate and think about money in your day-to-day life. This book is about Python and personal finance and how you can effectively mix the two together.

In *Personal Finance with Python* you will learn Python and finance at the same time by creating a profit calculator, a currency converter, an amortization schedule, a budget, a portfolio rebalancer, and a purchase forecaster. Many of the examples use pandas, the main data manipulation tool in Python. Each chapter is hands-on, self-contained, and motivated by fun and interesting examples.

Speakerdeck: /maxhumber (https://speakerdeck.com/maxhumber)

LinkedIn: /maxhumber (https://speakerdeck.com/maxhumber)

Twitter: @maxhumber (https://twitter.com/maxhumber)