Think Complexity

Introducing Allen Downey

Figure 1

Things you can think about with Allen Downey

- Think Python
- **Think Complexity**
- Think Stats
- Think Bayes
- Think DSP
- *Think Java*
- *Think Data Structures*

Allen Downey's Textbook manifesto

Students should read and understand textbooks.

- Always free
- Open source code
- Wikipedia reading assignments
- Problems to work on throughout

Complexity Science

Complexity science is the science of complex systems.

- Equations $\rightarrow$ simulations
- Analysis $\rightarrow$ computation
- Continuous $\rightarrow$ discrete
- Linear $\rightarrow$ non-linear
- Deterministic $\rightarrow$ stochastic

Chapters in Think Complexity

- Graphs
- **Small world graphs**
- Scale-free networks
- **Cellular automatons**
- **Game of Life**
- Physical modeling
- Self-organized criticality
- **Agent-based models**
- Herds, Flocks, and Traffic Jams
- Game Theory

Other reasons to read this book

- Data structures
- Algorithms
- Computational modeling
- Philosophy of science

My goals

- Introduce you to Allen Downey.
- Replicate some very famous experiments (Watts & Strogatz, Wolfram, Conway, Schelling).
- Teach you something new about Python.

Small world graphs

1. Simple graphs
2. Connected graphs
3. Random graphs
4. Regular graphs
5. Small world graphs

Complete graphs

```python
from itertools import combinations

def make_complete_graph(n):
    G = nx.Graph()
    nodes = range(n)
    G.add_nodes_from(nodes)
    G.add_edges_from(combinations(nodes, 2))
    return G

complete = make_complete_graph(10)
```

Creating edges for a regular graph

```python
def adjacent_edges(nodes, halfk):
    """Yields edges between each node and `halfk` neighbors
    n = len(nodes)
    for i, u in enumerate(nodes):
        for j in range(i+1, i+halfk+1):
            v = nodes[j % n]
            yield u, v
```

Create a Watts-Strogatz graph by rewiring edges

```python
def rewire(G, p):
    """Rewires each edge with probability `p`."""
    nodes = set(G.nodes())
    for edge in G.edges():
        if flip(p):
            u, v = edge
            choices = nodes - {u} - set(G[u])
            new_v = random.choice(tuple(choices))
            G.remove_edge(u, v)
            G.add_edge(u, new_v)
```

Cellular automatons

Popularized by Steven Wolfram in his book *A New Kind of Science*.

Calculating node clustering

```python
def node_clustering(G, u):
    """Computes local clustering coefficient for `u`."""
    neighbors = G[u]
    k = len(neighbors)
    if k < 2:
        return 0

    total = k * (k-1) / 2
    exist = 0
    for v, w in combinations(neighbors, 2):
        if G.has_edge(v, w):
            exist +=1
    return exist / total
```

Path length

```
def path_lengths(G):
    length_map = nx.shortest_path_length(G)
    lengths = [length_map[u][v] for u, v in itertools.combi
    return lengths
```

Cellular automatons

prev	111	110	101	100	011	010	001	000
next	0	0	1	1	0	0	1	0

Converting rules to tables

```python
def make_table(rule):
    """Make the table for a given CA rule."""
    rule = np.array([rule], dtype=np.uint8)
    table = np.unpackbits(rule)[::-1]  # ???
    return table

make_table(50)
```

Running a cellular automaton

```
cols = 11  # number of cells
rows = 5   # number of timesteps
array = np.zeros((rows, cols), dtype=np.int8)
array[0, cols//2] = 1  # turn center cell "on" at t0
table = make_table(50)

def step(array, i):
    """Executes one time step by computing the next row of
    corr = np.correlate(array[i-1], [4, 2, 1], mode='same')
    array[i] = table[corr]

for i in range(1, rows):
    step(array, i)
```

Game of Life

John Conway's Game of Life is a 2D cellular automaton!

For the vast implications of this simple work, see Daniel Dennett's books, *Consciousness Explained*, *Darwin's Dangerous Idea*, or *Freedom Evolves*.

Conway's Rule

current state	num neighbors	next state
live	2-3	live
live	0-1, 4-8	dead
dead	3	live
dead	0-2, 4-8	dead

Agent-based models

- Schelling's model of segregation
- Epstein & Axtell's Sugarscape