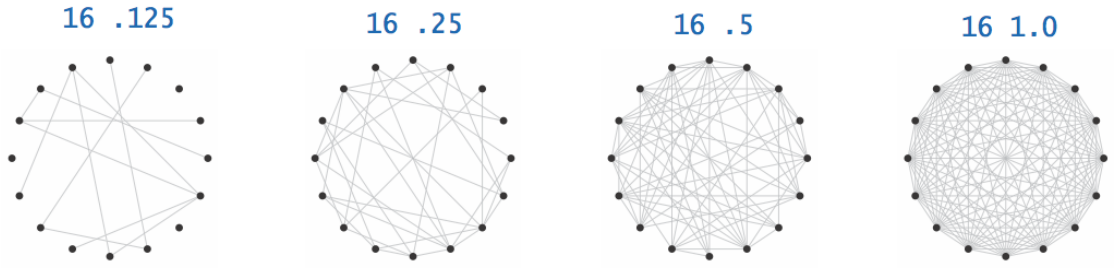
Computer Science E214

Tutorial 2

1 JBook 1.5.19

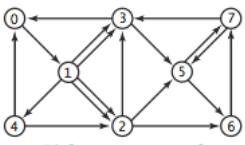
Write a program `randomgraph.py` that takes as command-line arguments an integer N and a floating-point value $p \in [0, 1]$. The program should plot N equally spaced points on the circumference of a unit circle. Each pair of points should furthermore be connected by a line with probability p, i.e. the presence of each line is determined independently with probability p.

Example output for various parameters is given below.



2 Discerning Surfing (JBook 1.6.2, 1.6.11, and 1.6.12)

Encode the eight-page example shown below for use as input to `transition.py` in a file called `eightpage.txt`. Then implement and use `transition.py` to obtain a transition matrix for the example. Use this matrix to calculate the page ranks by implementing and using both `randomsurfer.py` and `markov.py`.



Eight-page example

Now modify `transition.py` to ignore the effect of multiple links. That is, if there are multiple links from one page to another, count them as one link only. Call your modified program `singletransition.py`. Repeat the analysis above on the example, but using the modified transition matrix obtained with `singletransition.py`.

Repeat the analysis described above on the three-page example provided in `threepage.txt` in the tutorial resources.

Compare your results in each case: which do you think is a better approach to ranking web pages?

3 The binomial distribution (JBook 2.1.34)

Write a function

```
def binomial(n, k, p)
```

in a program `binomial.py` to compute the probability of obtaining exactly k heads in N flips of a biased coin (heads with probability p) using the formula

$$f(n,k,p) = p^k(1-p)^{n-k}N!/(k!(n-k)!)$$

Hint: To stave off overflow, compute $x = \ln f(n,k,p)$ and then return e^x. You may want to use an additional function for this.

Write a `main` function for your program that obtains N and p from the command line and prints out

$$\sum_{k=0}^{n} f(n,k,p) \ ,$$

which should be equal to one (up to numerical accuracy). Set up your program to call the main function when it is executed, but not when it is imported: in the former case, the value of the special variable `__name__` is `'__main__'`; in the latter, it is the name of the module (i.e. `'binomial'` in this case).

Also use the class `binomialclient.py` provided in the tutorial resources to test your `binomial` function. Take a look at the contents of `binomialclient.py` and ensure you understand how it tests the code.

4 The Erlang loss formula

Consider a model of a telephone network switch where λ is the arrival rate of telephone calls (measured in calls per second) to a switch that can carry maximally N calls, and $1/\mu$ is the average duration (measured in seconds) of a call. In telephony jargon, λ is called the *offered traffic* per unit time and $\rho = \lambda/\mu$ the *offered load*. The load ρ is measured in units of "Erlangs".

It is known that (under some reasonable assumptions we don't go into here) the probability that an incoming call is lost because all N circuits are busy is a function of the load, ρ, and N. This *blocking probability* is given by the *Erlang loss formula* (also known as the *Erlang B-formula*):

$$B(N,\rho) = \frac{\rho^N/N!}{\sum_{n=0}^{N}\rho^n/n!} \tag{1}$$

Eqn. (1) looks easy to evaluate, but think of the situation where $\rho = 100$ and $N = 200$. Then there is a term of the form $(100)^{200}/200!$ that needs to be evaluated, and this is difficult to do exactly, even for a computer. To get around this we use a recursive formulation for $B(N,\rho)$ which calculates $B(N,\rho)$ in terms of $B(N-1,\rho)$

$$B(N,\rho) = \frac{\rho\, B(N-1,\rho)}{N + \rho B(N-1,\rho)} \tag{2}$$

where $B(0,\rho) = 1$.

Create a program `erlang.py`, and use it to perform the following:

1. Write a recursive function `recErlang` which takes two parameters n and ρ and computes and returns the blocking probability $B(n,\rho)$ using Eqn. (2).

2. Determine the number of recursive calls that are used in total by this method to calculate the blocking probability — put your answer in the documentation for the method. (Note: If you are making more than one recursive call in your function, you are being inefficient, and you will run into trouble with large values of n — find a way to avoid it!)

3. Develop a non-recursive method `nonrecErlang` with the same parameters which also computes $B(n, \rho)$.

4. Implement a `main` function for testing that accepts n and ρ as command-line arguments, and then prints the results of the recursive and the non-recursive methods, as below:

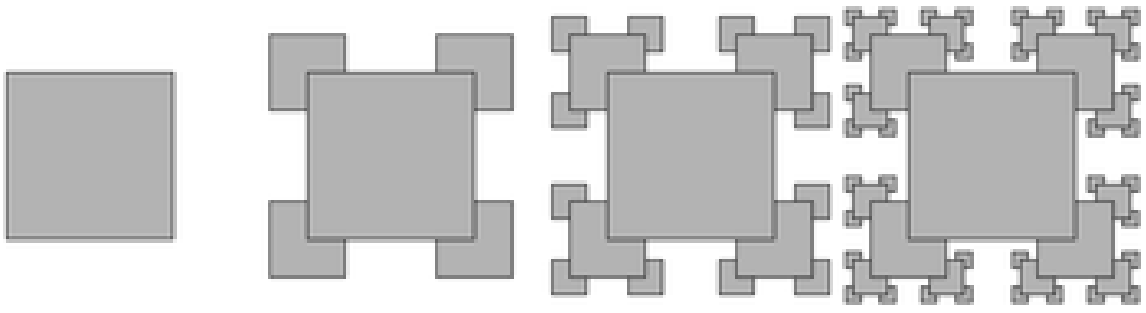
```
$ python erlang.py 10 20
Recursive: B(10, 20.000000) = 0.537963
Non-recursive: B(10, 20.000000) = 0.537963
```

5 Recursive Squares (JBook 2.3.22)

Write a program `recursivesquares.py` to produce each of the following recursive patterns shown in Figures 1 - 4, using `stddraw` to draw the squares. The ratio of the sizes of the squares is 2.2 : 1. To draw a shaded square, draw a filled gray square, then an unfilled black square. Give the program two command-line parameters: the first should choose between the different recursive patterns: 1 for *Front*, 2 for *Order*, 3 for *Right*, or 4 for *Shaded*; the second value n should specify the depth of the recursion. Figures 1 - 4 show the results for recursion depths of *one* through *four* (`(a)` - `(d)`). An example for running the program:

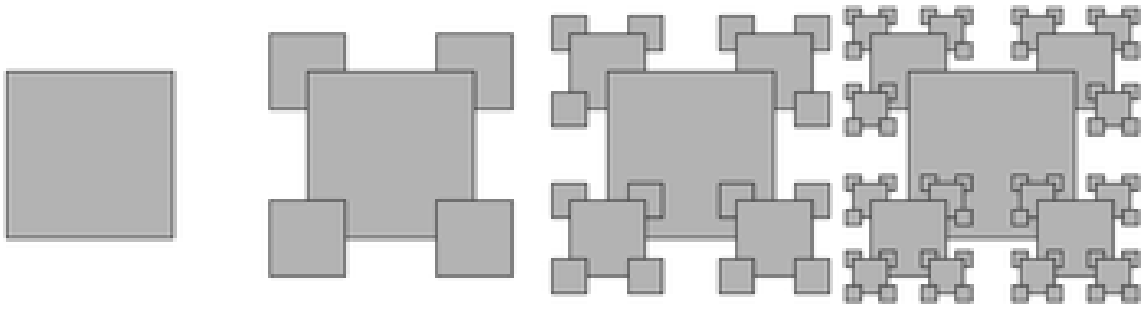
```
$ python recursivesquares.py 1 4
```

should render Figure 1d.



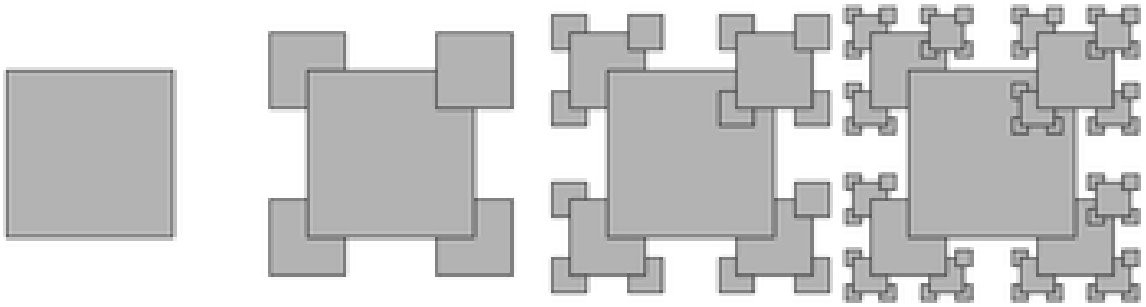
(a) (b) (c) (d)

Figure 1: Front



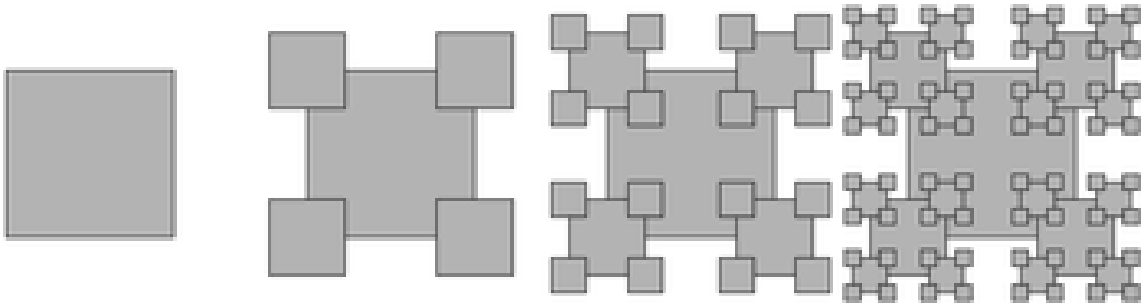
(a) (b) (c) (d)

Figure 2: Order



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

Figure 3: Right



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

Figure 4: Shaded