

1 Basics

Contents

1 Basics	1
1.1 Functions	1
1.2 Loops	1
1.3 Lists	1
1.4 Generating random numbers	2
1.5 Dynamical Systems	4
1.5.1 Investing	5
1.6 Plotting things	5
1.7 Noise: white and brown	6

The exercises below are intended to make sure that you have mastered the basics, e.g., writing functions, using loops, working with lists, etc., while doing something interesting.

1.1 Functions

1. Define a function `greet` such that `greet("John")` returns the string "Hello, John, how's it going?"
2. Use `math.log` and `math.exp` to define a function `power(a,k)` that raises a to the power k , where k may not be an integer.
3. Consider the sound waves produced by a piano playing middle C and the C one octave above. Each sound has a frequency. The standard middle C is 261.6 Hz. The middle C one octave above has a frequency twice as great, 523.2 Hz. The musical scale is divided into notes one half step apart (C to C-sharp, C-sharp to D, etc.). A half-step represents a frequency ratio R such that $R^{12} = 2$. Find this ratio. What kind of number is it?

1.2 Loops

1. Compute the sum $S_n = 1 - 1/2 + 1/3 - 1/4 \pm \dots \pm 1/n$ for $n = 10$ and $n = 100$. What is the significance of this sum?
2. Devise a program to compute the function $\zeta(s) = 1 + 1/2^s + 1/3^s + \dots$ using the partial sum $\zeta_n(s) = 1 + 1/2^s + 1/3^s + \dots + 1/n^s$. Compute values of $\zeta(1.5)$, $\zeta(2)$, and $\zeta(2.5)$ accurate to within 0.000001. How many terms were necessary in each case.

1.3 Lists

We give some examples which illustrate basic operations on lists, then give some exercises which use these operations.

Building lists

```
>>> foo = []                # empty list
>>> foo.append(2).          # append an element
>>> foo                     # check that it is there
[2]

>>> foo.append(3); foo.append(5)
>>> foo
[2, 3, 5]
```

Using a list as a stack

```
>>> foo.pop()
5
>>> foo
[2, 3]
```

Adding two lists

```
def add(a,b):
    outputList = []
    for i in range(len(a)):
        outputList.append(a[i] + b[i])
    return outputList

>>> add([1,2,3], [1,1,-1])
[2, 3, 2]

>>> add(["a", "b", "c"], ["=1", "=2", "=3"])
['a=1', 'b=2', 'c=3']
```

Exercises

1. Devise a function `uniquefy` that returns a list without repeats. Thus `uniquefy([1,1,3,3,1,2,3])` returns `[1,3,2]` and `uniquefy(['a', 'b', 'a', 'c', 'c'])` returns `['a', 'b', 'c']`.
2. Devise a function `isUnique` that returns `True` if a list has no repeats, false otherwise.

1.4 Generating random numbers

There are many uses in computer science and various applied fields for a source of random numbers, e.g., encryption. The best sources are physical. One can, for example, translate the clicks of a Geiger counter placed near a radioactive mineral into a random number generator. For a random number generator based on atmospheric noise, consult random.org.

Another approach is to use some kind of algorithm to generate sequences of numbers that have many of the same statistical properties as do true random number sequences. Because algorithms are deterministic, the sequences they produce are not truly random. We call them *pseudorandom*. We will describe the theory. Your task is to apply the theory to design and test some pseudorandom number generators.

Linear congruential generators

Choose integers a , b , and n . Consider the function

$$f(x) = (ax + b) \bmod n \quad (1)$$

The expression $y \bmod b$ computes the remainder of y upon division by n . Thus $17 \bmod 3 = 2$ and $18 \bmod 3 = 0$. In Python, we write these expressions as $17 \% 3$, etc. One can use f to generate a sequence of numbers as follows. First, choose a *seed* x_0 . Then, assuming that x_n has been computed, let $x_{n+1} = f(x_n)$.

Let's do an example with $f(x) = 5x + 1 \bmod 17$. First, we define f in Python:

```
$ python3
```

```
>>> def f(x):
...     return (5*x + 1) % 17
... 
```

Then we test it:

```
>>> f(0)
1
>>> f(1)
6
>>> f(6)
14
```

Finally, we use a loop to generate a sequence:

```

>>> a = 0
>>> for i in range(0,17):
...     print(a, end=' ')
...     a = f(a)
...
0 1 6 14 3 16 13 15 8 7 2 11 5 9 12 10 0

```

Exercises

1. Use the function f given above to generate pseudorandom numbers x in the unit interval. That is, x is a floating point number such that $0 \leq x < 1$.
2. Notice that there are no repeats in the sequence $\{x_n\}$ generated above. Does this happen in general? Experiment with various choices of a , b , and n . You may want to simplify and take $b = 0$.
3. Is there a way to automate the task of determining whether the numbers $\{x_n\}$, for $n = 0, 1, \dots, n-1$ has no repeats? That is, by applying some function to the output instead of eyeballing it.
4. Devise a function that returns a random element of a list.
5. Devise a function **nonsense(n)** that returns a nonsense string of length n . For example, **nonsense(5)** returns the string "axmuh" the first time it is called and "nqwof" the second time. (So it is not a function in the sense used in Mathematics.)
6. (Optional) Devise a function that returns a random sentence. The function should take as input several lists of words.
7. Construct a function **randomSuit()** which returns one of the strings "Heart", "Diamond", "Spade", "Club", each with equal probability.
8. (Optional) Construct a function **randomSuit(h, d, s, c)** which returns "Heart" with probability h , "Diamond" with probability d , etc. Of course $h + d + s + c = 1$.
9. (Optional, a small project) Construct a function **pokerHand** which returns a random five-card hand.

1.5 Dynamical Systems

A *dynamical system* consists of a set S of *states* and a function $f : S \rightarrow S$ that computes a new state from an old one. The set S may be the non-negative reals, the set of all reals, the set \mathbb{R}^n of real-valued n -vectors, the letters of the alphabet, the set of all alphanumeric strings, etc. If $x_0 \in S$ is the initial state of the system, then one defines a *trajectory* $\{x_n\}$ in S via the recursion relation $x_{n+1} = f(x_n)$.

Here is a simple example: S is the set of non-negative reals and $f(x) = kx$, where $k > 0$. This dynamical system models exponential growth (or decay). Here is a slightly more complicated model:

$$f(x) = kx(1 - x) \tag{2}$$

Exercises A

1. Compute the sequence $\{x_j\}$ where $x_0 = 0.5$, $j \leq N$, where $N = 10$, and where k takes on the values $0.5, 1.0, 1.5, \dots, 4.0$. Do you find anything of interest?
2. Repeat the preceding, but graph the results, but with significantly larger values of N . Do you find anything of interest?

1.5.1 Investing

Consider a parent who wants to save for a child's education. He/she sets an account upon the birth of the child with an initial amount of 1000 dollars. Annual return on the investment is 4 percent. Each year, 6000 dollars is added to the account. What will the value of the account be in eighteen years.

A problem like this is described by a dynamical system on the non-negative real numbers, $f : \mathbb{R}^+ \rightarrow \mathbb{R}^+$, where f has the form

$$f(x) = ax + b \tag{3}$$

Exercises B

1. Solve the investment problem above. Your solution should include a printout of the value of the account for each year.
2. Let's use the same dynamical system to solve a mortgage problem. A family buys a home, taking out a mortgage of 200,000 dollars at four percent annual interest. They make a monthly payment of 1000 dollars. How long will it take to pay off the mortgage? How much interest will the family have paid when the mortgage is paid off? Your solution should contain a month-by-month schedule of the family's mortgage balance.

1.6 Plotting things

The most basic way of using `matplotlib` is to construct two lists, the x-values and the y-values. Call these lists `xs` and `ys`. Putting them together gives a list of points in the plane. Joining successive pairs of points constructs a figure. That is the figure drawn by `plt.plot(xs, ys)`. The `xs` and `ys` can be either lists of numbers or numpy arrays.

1. Let `xs = [0,1,1,0]`, `ys = [0, 0, 1, 1]`. What figure will `plt.plot(xs, ys)` make? Answer using pure thought, then use Python to check your answer.
2. Draw a square using `matplotlib`.

Fourier series

In class we used `matplotlib` to graph the Fourier polynomial

$$f(t) = \sin \pi t + \frac{1}{3} \sin 3\pi t + \frac{1}{5} \sin 5\pi t \quad (4)$$

This function is one element of the family of functions

$$f_n(t) = \sin \pi t + \frac{1}{3} \sin 3\pi t + \cdots + \frac{1}{2n+1} \sin(2n+1)\pi t \quad (5)$$

where the sum is taken over odd integers. Thus the function in equation (4) is f_2 . Plot f_3 and f_4 .

Exercise A. What is the limit of the graph of f_n as n tends to infinity?

Note. Consider the Fourier series

$$f(t) = a_0 + \sum_{n=1}^{\infty} a_n \cos n\omega t + \sum_{n=1}^{\infty} b_n \sin n\omega t \quad (6)$$

The coefficients a_i and b_i are called the *amplitudes*. Let us suppose that $f(t)$ represents the voltage in a circuit. As we know from basic physics, the current in a circuit is governed by Ohm's law, $V = IR$. Moreover, power is given by voltage times current, $P = VI$, so that finally $P = V^2/R$. In other words, power is proportional to the square of the voltage, where the constant of proportionality is the resistance. Returning to the Fourier series $f(t)$, we find that the instantaneous power in mode n is $a_n^2 + b_n^2$. If the series is written entirely in terms of sines (as it may be using a phase shift), then the instantaneous power is simply b_n^2 . In conclusion: *power per mode is given by the square of the Fourier coefficients*. The sequence $\{b_n^2\}$ is called the *power spectrum*.

Exercise B. Graph (perhaps by hand) the power spectrum of f_4 .

1.7 Noise: white and brown

White noise is produced by uniformly distributed voltage fluctuations. It can be modeled like this:

```
import random as r
```

```
ys = []
for i in range(0,100):
    ys.append(r.random())
```

And it can be graphed like this:

```
xs = list(range(100))

plt.plot(xs, ys)
```

Exercise A.. Plot white noise

Brown noise is produced by a different kind of random voltage fluctuation: begin with some initial voltage, say 0. Then add successive uniformly distributed random numbers, say, in the range -0.1 to $+0.1$. We can generate brown noise like this:

```
ys = [0]
for i in range(0,99):
    ys.append(ys[i] + 0.1*(r.random() - 0.5))
```

Exercise B. Plot brown noise

Exercise C. (1) Listen to both white and brown noise; (2) investigate physical sources of both types of noise. (3) Which type of noise do you prefer?

Note. We will discuss the power spectra of white and brown noise in class and learn how to compute spectra using the `numpy` package.

References for noise

[Listen to white noise](#)

[Listen to brownian noise](#)

[About white noise \(Wikipedia\)](#)

[About brownian noise \(Wikipedia\)](#)