

CPSC 217 Exercises

John Aycock

W2011

TRY THE EXERCISES YOURSELF before looking at the solutions; computer programming is like performing a musical instrument, and while you can learn things from studying other people's approach, you learn the most by trying it yourself. Note also that there are many ways of solving these exercises, and so long as your solution works properly, that's all that matters at this point. Different solutions have different tradeoffs, however – see if you can identify why you might prefer one solution over another.

1 Difficulty 1* (Easiest)

1. Print a string out forwards, one character at a time.

Solution.

```
s = 'abcde'
i = 0
while i < len(s):
    # or print(s[i], end='') to be fancy
    print(s[i])
    i = i + 1
```

Solution.

```
s = 'abcde'
for ch in s:
    print(ch)
```

Solution.

```
s = 'abcde'
for i in range(len(s)):
    print(s[i])
```

2. Print a string out backwards, one character at a time.

Solution.

```
s = 'abcde'
i = 1
while i <= len(s):
    print(s[-i])
    i = i + 1
```

Solution.

```
s = 'abcde'
i = len(s) - 1
while i >= 0:
    print(s[i])
    i = i - 1
```

Solution.

```
# advanced use of the built-in range function
s = 'abcde'
for i in range(len(s)-1, -1, -1):
    print(s[i])
```

3. Print the arithmetic mean (a.k.a. the “average”) of a list of numbers.

Solution.

```
L = [1, 2, 3, 4, 5, 6, 7, 8]
sum = 0
for n in L:
    sum = sum + n
print(sum / len(L))
```

4. Split a string into words and print out the first letter of each word.

Solution.

```
s = 'The quick brown fox jumps over the lazy dog.'
words = s.split(' ')
for word in words:
    print(word[0])
```

5. Read lines of input until a sentinel of -1 is read, printing out each line with its corresponding line number beside it.

Solution.

```
n = 1
while True:
    line = input()
    if line == '-1':
        break
    print(n, line)
    n = n + 1
```

6. Read lines of input until a sentinel of -1 is read, then print out the lines in reverse order.

Solution.

```

lines = []
while True:
    line = input()
    if line == '-1':
        break
    lines.append(line)

i = len(lines)-1
while i >= 0:
    print(lines[i])
    i = i - 1

```

Solution.

```

lines = []
while True:
    line = input()
    if line == '-1':
        break
    lines.append(line)

# use list's built-in reverse function
lines.reverse()

for line in lines:
    print(line)

```

Solution.

```

lines = []
while True:
    line = input()
    if line == '-1':
        break
    # insert into list in backwards order
    # to begin with, then print out
    lines.insert(0, line)

for line in lines:
    print(line)

```

2 Difficulty 2*

1. Print the Fibonacci sequence: 0, 1, 1, 2, 3, 5, 8, 13, 21, ... Stop once you've exceeded 100.

Solution.

```

n_2 = 0
n_1 = 1

# these don't get printed in the loop

```

```

print(0)
print(1)

while True:
    n = n_2 + n_1
    if n > 100:
        break
    print(n)
    n_2 = n_1
    n_1 = n

```

Solution.

```

# computes the nth Fibonacci number recursively
def fib(n):
    if n == 0:
        return 0
    if n == 1:
        return 1
    return fib(n-2) + fib(n-1)

i = 0
while True:
    n = fib(i)
    if n > 100:
        break
    print(n)
    i = i + 1

```

2. Read lines of input until the sentinel `%%EOF` is seen. Print out all lines unless they are between a `%%HIDE` line and a `%%SHOW` line. For example, the input

```

foo bar
baz
%%HIDE
blarg
%%SHOW
garble
%%HIDE
greet qwerty
uiop
%%SHOW
the penultimate line
%%EOF

```

would result in

```

foo bar
baz
garble
the penultimate line

```

For bizarre reasons, the file containing these exercises is actually run through a Python script that does exactly this!

Solution.

```
EOF = '%%EOF'
HIDE = '%%HIDE'
NOHIDE = '%%SHOW'

hiding = False
while True:
    line = input()
    if line == EOF:
        break
    if hiding:
        if line == NOHIDE:
            hiding = False
    else:
        if line == HIDE:
            hiding = True
        else:
            print(line)
```

3. Draw the Mandelbrot set using Python's turtle graphics module. This is a slightly different challenge as there is pseudocode available on the Wikipedia page describing the Mandelbrot set:

http://en.wikipedia.org/w/index.php?title=Mandelbrot_set&oldid=343535421

Can you convert it into Python code?

3 Difficulty 3*

The solutions to these are so varied that I'm not going to include answers. If you can do these and want to discuss your solution and possible alternatives, please come see me! Note that you'll probably need to do a bit of research on these problems first.

1. The traditional mapping of letter to keys on a cell phone keypad is, to be polite, nonoptimal for sending text messages. When adjacent letters in the message are on the same key, you have to either wait for the cursor to advance or press a button to forward the cursor yourself – let's call both ways to advance the cursor *skips*.

Write a program which takes as input a message (alphabetic characters only) and a number of keys. The program's output should be a mapping of letters to keys that tries to minimize the number the number of skips required to send that text message. It should also print the number of skips required with the traditional keypad layout, as well as the number of skips for your new layout.

Hint (especially for math majors): you can represent this as a graph coloring problem.

2. Steganography refers to the ability to hide messages. Write a program to embed a message into an image that's in PPM ASCII format (magic number P3). Break the message into individual bits and put one bit in each pixel's RGB color value (i.e., three bits of hidden message per pixel). Put each message bit into the LSB (least significant bit) of the color value. Note that if you implement this correctly, the presence of the hidden message in the picture will not be visually obvious. Hint: Python's bitwise AND operator (&) can be used to set a bit to zero; the bitwise OR operator (|) can be used to set a bit to one.

Of course, you should also write a program to extract the hidden message afterwards!