1. (Problem 7 from Section 2.10 in D. Bindel's and J. Goodman's book "Principles of Scientific Computing") Starting with the declarations

```
      float x, y, z, w;
const float oneThird = 1/ (float) 3;
const float oneHalf = 1/ (float) 2;
// const means these never are reassigned
```

we do lots of arithmetic on the variables `x`, `y`, `z`, `w`. In each case below, determine whether the two arithmetic expressions result in the same floating point number (down to the last bit) as long as no `NaN` or `inf` values or denormalized numbers are produced.

(a) `( x * y ) + ( z - w )`
`( z - w ) + ( y * x )`

> **Solution:** Yes, these expressions result in the same floating point number; as mentioned in Section 2.2.2, floating point addition and multiplication are commutative.

(b) `( x + y ) + z`
`x + ( y + z )`

> **Solution:** No, these expressions do not necessarily result in the same floating point number. This is demonstrated by the following C code:
>
> ```
> bool is_nonassociative(float x, float y, float z) {
>   float lhs = (x + y) + z;
>   float rhs = x + (y + z);
>   printf("lhs=%.8f\nrhs=%.8f\n", lhs, rhs);
>   return lhs != rhs;
> }
>
> int main() {
>   // Addition isn't associative
>   assert(is_nonassociative(1.0, 0.0000002, 0.0000003));
>   return 0;
> }
> ```

(c) `x * oneHalf + y * oneHalf`
`( x + y ) * oneHalf`

> **Solution:** Yes, these expressions result in the same floating point number; consider the floating point representations of `x`, `y`:
>
> - `x` $= (-1)^s \cdot (1.b_1 b_2 \dots b_d) \cdot 2^e$
> - `y` $= (-1)^{s'} \cdot (1.b'_1 b'_2 \dots b'_d) \cdot 2^{e'}$
>
> Multiplication by `oneHalf` is done exactly by decrementing the exponent, and hence
>
> - `x * oneHalf` $= (-1)^s \cdot (1.b_1 b_2 \dots b_d) \cdot 2^{e-1}$
> - `y * oneHalf` $= (-1)^{s'} \cdot (1.b'_1 b'_2 \dots b'_d) \cdot 2^{e'-1}$

Without loss of generality, assume $e < e'$, so

$$\texttt{x * oneHalf + y * oneHalf} = 2^{e-1}\left((-1)^s \cdot (1.b_1b_2\ldots b_d) + (-1)^{s'} \cdot (1.b'_1b'_2\ldots b'_d) \cdot 2^{e'-e}\right).$$

On the other hand,

$$\texttt{x + y} = 2^{e}\left((-1)^s \cdot (1.b_1b_2\ldots b_d) + (-1)^{s'} \cdot (1.b'_1b'_2\ldots b'_d) \cdot 2^{e'-e}\right)$$

and so multiplying by `oneHalf` is done exactly by decrementing the exponent to yield the same result as the other expression.

(d) `x * oneThird + y * oneThird`
`( x + y ) * oneThird`

Solution: No, these expressions do not necessarily result in the same floating point number. This is demonstrated by the following C code:

```
bool is_nondistributive(float x, float y, float z) {
  float lhs = x * z + y * z;
  float rhs = (x + y) * z;
  printf("lhs=%.8f\nrhs=%.8f\n", lhs, rhs);
  return lhs != rhs;
}

int main() {
  // Multiplication isn't distributive
  const float oneThird = 1 / (float)3;
  assert(is_nondistributive(1.000003, 0.000002, oneThird));
  return 0;
}
```

2. The *tent map* of the interval $[0, 1]$ onto itself is defined as

$$f(x) = \begin{cases} 2x, & x \in [0,1/2) \\ 2 - 2x, & x \in [1/2, 1]. \end{cases} \tag{1}$$

Consider the iteration $x_{n+1} = f(x_n)$, $n = 0, 1, 2, \ldots$.

(a) What are the fixed points of this iteration, i.e. the points x^* such that $x^* = f(x^*)$? Show that these fixed points are unstable, i.e., if you start iteration at $x^* + \delta$ for any δ small enough then the next iterate will be farther away from x^* then $x^* + \delta$.

Solution: Suppose $x^* \in [0, 1]$ is such that $x^* = f(x^*)$. It is either the case that $x^* \in [0,1/2)$ or $x^* \in [1/2, 1]$. If $x^* \in [0,1/2)$, then $x^* = 2x^*$, so $x^* = 0$. Otherwise, $x^* = 2 - 2x^*$, so $x^* = 2/3$. Hence, the fixed points of this iteration are 0 and $2/3$.

We will now show that these fixed points are unstable; consider first the case of $x^* = 0$.

Then for $0 < \delta < 1/2$,

$$f(x^* + \delta) = f(\delta) = 2\delta$$

which is clearly further from 0 than δ, so 0 is unstable.

Now consider $x^* = 2/3$. For $-1/6 \leq \delta \leq 1/3$, then $x^* + \delta \in [1/2, 1]$, so

$$f(x^* + \delta) = f(2/3 + \delta) = 2 - 2(2/3 + \delta) = 2/3 - 2\delta$$

which again is clearly further from $2/3$ than $2/3 + \delta$, so $2/3$ is also unstable.

(b) Prove that if x_0 is rational, then the sequence of iterates generated starting from x_0 is periodic.

Solution: Let $x_0 \in [0, 1]$ be rational, in which case there exist $a_0, b \in \mathbb{N}$ such that $x_0 = a_0/b$. We will prove by induction that every x_n can be written as a_n/b with $a_n \in \mathbb{N}$ such that $0 \leq a_n \leq b$.

As the base case, we have that $0 \leq x_0 \leq 1$. Rewriting x_0 as a_0/b and multiplying by b yields $0 \leq a_0 \leq b$, and hence the claim holds for $n = 0$.

As the inductive step, suppose x_n can be written as a_n/b with $0 \leq a_n \leq b$. Then note

$$x_{n+1} = f(x_n) = \begin{cases} 2x_n, & x_n \in [0, 1/2) \\ 2 - 2x_n, & x_n \in [1/2, 1] \end{cases} = \begin{cases} 2a_n/b, & a_n \in [0, b/2) \\ 2(b - a_n)/b, & a_n \in [b/2, b] \end{cases}$$

and hence

$$a_{n+1} = \begin{cases} 2a_n, & a_n \in [0, b/2) \\ 2(b - a_n), & a_n \in [b/2, b] \end{cases}$$

In the first case, $0 \leq a_n < b/2$ and so multiplying by 2 yields that $0 \leq 2a_n = a_{n+1} < b$. In the second case, $b/2 \leq a_n \leq b$ and so multiplying by -1 yields

$$-b \leq -a_n \leq -b/2.$$

Then adding b yields

$$0 \leq b - a_n \leq b/2$$

and finally multiplying by 2,

$$0 \leq 2(b - a_n) = a_{n+1} \leq b$$

so the claim holds in both cases, and hence x_{n+1} can be written as a_{n+1}/b with $0 \leq a_{n+1} \leq b$. By the principle of induction, the claim holds for all $n \in \mathbb{N}$.

Finally, note that there are only finitely many values a_n such that $0 \leq a_n \leq b$; in particular, $b + 1$ many. Hence, it must be that the sequence of iterates is periodic, as it can take only finitely many values.

(c) Show that for any period length p, one can find a rational number x_0 such that the

sequence of iterates generated starting from x_0 is periodic of period p.

Solution: Let $p \in \mathbb{N}$. We claim that $x_0 \coloneqq \frac{2}{2^p+1}$ generates a periodic sequence of iterates of period p. By part (b) the sequence of iterates must be periodic.

To see that this sequence has period p, note that

$$x_0 = \frac{2}{2^p+1} < \frac{2^2}{2^p+1} < \cdots < \frac{2^{p-2}}{2^p+1} < \frac{2^{p-1}}{2^p+1} < \frac{1}{2} < \frac{2^p}{2^p+1}$$

and hence the sequence will double until reaching $x_{p-1} = \frac{2^p}{2^p+1}$. However,

$$f(x_{p-1}) = 2 - 2x_{p-1} = \frac{2}{2^p+1} = x_0$$

so the sequence of iterates is periodic with terms $\{x_0, x_1, x_2, \ldots, x_{p-2}, x_{p-1}\}$, and hence is of period p.

(d) Generate several long enough sequences of iterates on a computer using any suitable language (Matlab, Python, C, etc.) starting from a pseudorandom x_0 uniformly distributed on $[0, 1)$ to observe a pattern. I checked in Python and C that 100 iterates are enough. Report what you observe. If possible, experiment with single precision and double precision.

Solution: From the Python script below, we find that after about 53 iterations, the sequence reaches 1, and then $f(1) = 0$. Finally, since $f(0) = 0$, the sequence remains 0 thereafter.

```
import random

# Tent map on the interval [0, 1]
def f(x):
  return (2 * x) if (x < 0.5) else 2 - (2 * x)

def iterate(n):
  x = random.random()
  print(x)
  for i in range(n):
    x = f(x)
    print(x)

iterate(55)
```

(e) Explain the observed behavior of the generated sequences of iterates.

Solution: One bit of precision is lost at each iteration by multiplying by 2 until eventually all precision is lost.