1. .

   a. .

   b. For n=2, it took 43,022 nanoseconds. For n=3: 3,918,102 nanoseconds.

   c. .

   d.

n	50	500	5000
abundant_density(n)	0.18	0.242	0.2478

   We can see that as n gets bigger, the density does indeed start to fall closer to the predicted range - while 50 and 500 are still out of the range, for 5000 the result falls within it.

   e. .

   f. If 6|n:

   That means that 2 and 3 divisors of n too, thus n/2 and n/3 are also divisors, so: n/2+n/3+n/6 = (18+12+6)n/36 = n and n is semi perfect by the third order.

   If n is semi perfect by the third order:

   So (similar to the proof in Q1e in the code) it must be an even number, meaning the sum must consist of n/2. Let's falsely assume that n is not divisible by 3, so the highest possible sum is: n/2+n/4+n/5 = (20+10+8)n/40 = 0.95n which is contradictory to the fact that n is semi perfect by the third order. So, n is divisible by both 2 and 3, meaning it is divisible by 6.

2. .

   a. .

   b. .

   c. .

   d. To answer this question, I wrote the following line of code:

```
sum([roulette_repeat(60, 10) for _ in range(200)]) / 200
```

Thus, I've tested an average of 200 games of the roulette for 10 repeats, and almost every time got a positive value, meaning the user gained money.

However, when I increased the roulette repeat by a lot (to 500 repeats):

```
sum([roulette_repeat(60, 500) for _ in range(200)]) / 200
```

The result was always negative, and the player lost more than 10 times their original bet.

This kind of makes sense, as also in the real world gamblers usually make "easy" money while playing only a few rounds, and those that continue playing usually end up losing everything.

e. .

3. .

a. .

b. .

c. .

d. .

e. .

f. .

g. Let's mark n as the amount of digits of the number in base c. So, as we saw in the recitation, if N is the number in base 10, the following is true: $n = [\log_c N]$. But we also know that N has d digits in base b, so the upper limit for N is b^d, meaning the following is true: $N <= b^d \Rightarrow [\log_c N] <= [\log_c (b^d)] \Rightarrow n <= [d * \log_c b]$.

P.S. here [x] means the same as ceil(x).