1) a) As 9199 is prime and $9199 \nmid 1998$ we have with FLT

$$1998^{9198} = 1998^{9199-1} \equiv 1 \pmod{9199}.$$

b) With Bézout's lemma we compute the following:

(1)	$991 =$	$0 \cdot 64+$	$1 \cdot 991$	
(2)	$64 =$	$1 \cdot 64+$	$0 \cdot 991$	
(3)	$31 =$	$(-15) \cdot 64+$	$1 \cdot 991$	$(1) - 15 \cdot (2)$
(4)	$2 =$	$31 \cdot 64+$	$(-2) \cdot 991$	$(2) - 2 \cdot (3)$
(5)	$1 =$	$(-480) \cdot 64+$	$31 \cdot 991$	$(3) - 15 \cdot (4)$

So according to Lemma on slide 15 of week 9 the inverse $\overline{64}^{-1} = \overline{-480}$. Due to

$$-480 + 1 \cdot 991 = 511$$

the inverse of 64 mod 991 is 511.

Since $\gcd(64, 2) = 2$ and $\gcd(64, 4) = 4$ the congruence class of 64 does not have an inverse mod n for $n = 2, 4$.

Alternatively, FLT can be employed to obtain that $64 \cdot (64^{989}) = 64^{990} \equiv 1 \pmod{991}$. Computing 64^{989} mod 991 yields 511.

c) Since $\gcd(11, 14) = 1$ and $14 = 7 \cdot 2$ we know by Euler's Theorem that $11^6 = 11^{(7-1)\cdot(2-1)} \equiv 1 \pmod{14}$. Thus, $11^{182} = 11^{6 \cdot 30+2} = (11^6)^{30} \cdot 11^2 \equiv 1^{30} \cdot 11^2 \equiv 121 \equiv 9 \pmod{14}$.

2) a) We have to justify that 1064984 is the outcome of each of the 4 ways of computing.

The value of $123456789^{987654321}$ requested in the first two ways is rather large, approximately 10^{10} digits ($8 \cdot 10^9$), and because of this computing the value *is not* immediately feasible, irrespectively of how that is done.

Hence, instead we first verify the third way, checking that `expmod 123456789 987654321 1678321` evaluates to `1064984` in Haskell,[1] with `expmod` as on slide, which *is* feasible because the intermediate results involved have at most 7 digits (the number of digits of 1678321).

From this then, the result of the first two ways follows, using the properties of modular arithmetic (that taking results modulo at intermediate stages does not change the result, modulo as discussed in the lecture)[2] and correctness of fast exponentiation (that it produces the same result as exponentiation; see the slides).

Finally, correctness of the fourth way follows from FLT, using that 1678321 is prime to conclude that $a^{1678320} \equiv 1 \pmod{1678321}$ so that $a^{987654321} \equiv a^{987654321 \bmod 1678320} \equiv a^{802161} \pmod{1678321}$, using as before modular arithmetic, fast exponentiation, and Haskell to evaluate `expmod 939356 802161 1678321`.

[1] Implicitly this relies on Haskell computations on `Integer`s to be correct, so implicitly on correctness of the (default) algorithms used for that from GMP. We also rely on `expmod` being a correct implementation of fast exponentiation modulo.

[2] More precisely, $(x \bmod m) \cdot (y \bmod m) \equiv x \cdot y \pmod{m}$, and *mutatis mutandis* the same for addition, subtraction, and exponentiation.

b) The four ways are already ordered in decreasing time (and space) complexity. This can be verified by timing the respective computations.[3]

Alternatively, we can estimate the number of 'operations' each requires and conclude from that. For instance, we may count the number of *multiplications* of each way. These numbers are approximately 10^9 for the first way (repeated multiplication), 30 ($9 \cdot \log_2 10$) for the second and third ways, and 20 ($6 \cdot \log_2 10$) for the fourth.

Since we already observed that there's a large difference between the second and third ways in the time their actual computation takes, it is clear that just counting the number of multiplications of *numbers*, so assuming each multiplication takes the same amount of time, is a bit simplistic; the more digits numbers have the more time it takes multiplying them. It seems more reasonable to count the multiplication of an m-digit number by an n-digit number as $m \cdot n$, i.e. to count the number of multiplications of *digits*.[4] Then the respective counts are approximately 10^{20} ($\sum_{i=1}^{10^9}(i \cdot 9) \cdot 9$, as multiplication by 10^9 makes a number 9 digits longer, $9 \cdot 9 \approx 10^2$ and $\frac{n \cdot (n+1)}{2} = \sum_{i=1}^{n} i$) for the first way, $\frac{1}{4} \cdot 10^{20}$ ($(\frac{1}{2} \cdot 10^{10})^2$ based on assuming that the last operation to get to our result of 10^{10} digits, was squaring of a number having half its digits ($\frac{1}{2} \cdot 10^{10}$), and that that is the dominant factor) for the second, $\frac{3}{2} \cdot 10^3$ ($30 \cdot 7^2$ for 30 multiplications of two 7-digit numbers) for the third, and 10^3 ($20 \cdot 7^2$) for the fourth way. Note that now, in accordance with the experiments, the complexity of the third is way smaller than that of the second way.

3) a The situation described in the exercise corresponds to having three modulo equations, namely $x \equiv 2 \pmod 4$, $x \equiv 2 \pmod 5$, and $x \equiv 3 \pmod 7$, where x is the number of Lego bricks. Given that $0 \leq x < 200$ we have to find *the* number x, i.e. we have to find an x satisfying these constraints and show that it is the unique such.

We proceed by first solving the first *two* congruences, i.e. by transforming it into a *single* new one (ad presented in the lecture), and then do that once more to solve the combination of that new one with the *third*.

Consider the first two congruences $x \equiv 2 \pmod 4$ and $x \equiv 2 \pmod 5$. By the Chinese Remainder Theorem, which may be applied since $\gcd(4, 5) = 1$, the solutions to these *two* congruences are the *same* as the solutions of the *single* congruence $x \equiv 2 \pmod{4 \cdot 5}$, where the 2 in the congruence is computed as $v \cdot q \cdot a + u \cdot p \cdot b$ after setting $a = 2$, $p = 4$, $b = 2$, $q = 5$ in Bézout's Lemma and computing u and v. That yields the solutions $u = -1$ and $v = 1$ of $\gcd(4, 5) = 1 = u \cdot 4 + v \cdot 5$, from which we indeed find $v \cdot q \cdot a + u \cdot p \cdot b = 1 \cdot 5 \cdot 2 + (-1) \cdot 4 \cdot 2 = 2$.

Now we are left with solving both the new congruence $x \equiv 2 \pmod{20}$ and the third one $x \equiv 3 \pmod 7$. Again we apply the Chinese Remainder Theorem, using that $\gcd(20, 7) = 1$, so the solutions to these two congruences are the same as the solutions to the single congruence $x \equiv 122 \pmod{20 \cdot 7}$, where the 122 is computed as $v \cdot q \cdot a + u \cdot p \cdot b$ now by taking $a = 2$, $p = 20$, $b = 3$, $q = 7$ in Bézout's Lemma, again computing u and v. That yields solutions $u = -1$ and $v = 3$ of $\gcd(20, 7) = 1 = u \cdot 20 + v \cdot 7$, from which we indeed find $v \cdot q \cdot a + u \cdot p \cdot b = 3 \cdot 7 \cdot 2 + (-1) \cdot 20 \cdot 3 = -18 \equiv 122 \pmod{140}$.

Thus the solutions to the initial three congruences are the *same* as the solutions to $x \equiv 122 \pmod{140}$. Since 122 is the only such satisfying the further constraint $0 \leq x < 200$,

[3] Although comparing the first two may be inconclusive due to infeasibility.

[4] To see this is not unreasonable try multiplying numbers having many digits 'by hand'.

we conclude that 122 is the *one and only* solution.[5]

Alternatively, we can 'brute-force it' by searching for a number x having the required properties. A list having 122 as its *one and only* element is obtained by, in Haskell, evaluating

```
[ x | x <- [0..199], x `mod` 4 == 2, x `mod` 5 == 2, x `mod` 7 == 3]
```

b A message m encrypted by RSA into a code c, can be decrypted by computing $c^d \mod p \cdot q$, for d the private key and public key $(e, p \cdot q)$. However, since p and q are relatively prime, there is an alternative way (as discussed in the lecture, with code) based on the CRT to obtain m, namely by first computing a as $c^{d \mod (p-1)} \mod p$ and b as $c^{d \mod (q-1)} \mod q$, and then combining them to compute m as $a + p \cdot ((p' \cdot (b - a)) \mod q) \mod (p \cdot q)$.[6]

The reason for preferring to compute m like this, is that it is faster: We do exponentiations modulo p and q separately, instead of the more expensive exponentiation modulo $p \cdot q$ (four times more expensive as intermediate results typically have twice the number of digits, assuming p and q have roughly the same number of digits). Moreover, both exponents $d \mod (p-1)$ and $d \mod (q-1)$ are smaller than the exponent d.

4*) We have the ciphertext c and the public key (e, n). We do a prime factorization of n (taken from `https://stackoverflow.com/questions/21276844`) subtract 1 from each prime and multiply the results again, which gives us ϕ. We obtain the private key d using `invmod` from the slides calculating the inverse of $e \mod \phi$. Finally, we only have to decrypt the message either with `decrypt` or with `expmod` followed by `decode`. An example of Haskell code is given below:

```
primefac :: Integer -> [Integer] -- factorize the integer n into its prime factors
primefac 1 = []
primefac n
  | factors == []  = [n]
  | otherwise = factors ++ primefac (n `div` (head factors))
  where factors = take 1 $ filter (\x -> (n `mod` x) == 0) [2 .. n-1]

rsa_attack :: Integer -> (Integer,Integer) -> Integer -- attack a given ciphertext
-- c using the public key (n,e)
rsa_attack c (e,n) = expmod c d n where
    phi = product (map (subtract 1) (primefac n))
    d = invmod e phi
```

[5] Many people gave alternative ways for arriving at 122 as *a* solution. That is only 'half' the answer as that leaves open the possibility there are other solutions. That is, a *proof* is needed that 122 is the *only* solution (and that proof was usually missing; in the above we relied on the CRT for such a proof). Correspondingly, also in a brute-force method we cannot stop our search after finding *a* solution, as some people did, but have to exhaust all numbers x in the interval $0 \leq x < 200$ to conclude that it is the *unique* solution.

[6] Note that although in principle, encryption could be sped up in this way as well, RSA is based on public key encryption so on the public not knowing the private key: if the public were to *know* both p and q on top of the public key $(e, p \cdot q)$, there would be no point in encrypting at all, since then it could compute the private key d, as the inverse of e modulo $(p-1) \cdot (q-1)$, and then easily decrypt all messages.