MATH182 HOMEWORK #1
DUE June 18, 2020

Note: while you are encouraged to work together on these problems with your classmates, your final work should be written in your own words and not copied verbatim from somewhere else. You need to do at least seven (7) of these problems. All problems will be graded, although the score for the homework will be capped at $N :=$ (point value of one problem) $\times$ 7 and the homework will be counted out of N total points. Thus doing more problems can only help your homework score. For the programming exercise you should submit the final answer (a number) *and* your program source code.

Exercise 1. *Write out the following two sums in full:*

(1) $\sum_{0\leq k\leq 5} a_k$
(2) $\sum_{0\leq k^2\leq 5} a_{k^2}$

Solution. (1) First we need to determine the set $\{k \in \mathbb{Z} : 0 \leq k \leq 5\}$. This is $\{0, 1, 2, 3, 4, 5\}$. Thus these are the indices of the terms which appear in the sum, so:

$$\sum_{0\leq k\leq 5} a_k \;=\; a_0 + a_1 + a_2 + a_3 + a_4 + a_5.$$

(2) First we need to determine the set $\{k \in \mathbb{Z} : 0 \leq k^2 \leq 5\}$. This is $\{-2, -1, 0, 1, 2\}$. Thus these are the indices k we are summing over, so

$$\sum_{0\leq k^2\leq 5} a_{k^2} \;=\; a_{(-2)^2} + a_{(-1)^2} + a_{0^2} + a_{1^2} + a_{2^2} \;=\; a_0 + 2a_1 + 2a_4. \qquad \square$$

Exercise 2. *Evaluate the following summation:*

$$\sum_{k=1}^{n} k2^k.$$

Hint: rewrite as a double sum.

Solution. Note that

$$\begin{aligned}
\sum_{k=1}^{n} k2^k &= \sum_{k=1}^{n}\sum_{j=1}^{k} 2^k \\
&= \sum_{1\le j\le k\le n} 2^k \\
&= \sum_{j=1}^{n}\sum_{k=j}^{n} 2^k \quad \text{summing over } k \text{ first} \\
&= \sum_{j=1}^{n}\left(\sum_{k=0}^{n} 2^k - \sum_{k=0}^{j-1} 2^k\right) \quad \text{manipulating the domain} \\
&= \sum_{j=1}^{n}\left(\frac{1-2^{n+1}}{1-2} - \frac{1-2^j}{1-2}\right) \quad \text{geometric sums} \\
&= \sum_{j=1}^{n}\left(2^{n+1} - 1 - 2^j + 1\right) \\
&= \sum_{j=1}^{n}\left(2^{n+1} - 2^j\right) \\
&= \sum_{j=1}^{n} 2^{n+1} - \sum_{j=1}^{n} 2^j \\
&= n2^{n+1} - \left(\frac{1-2^{n+1}}{1-2} - 1\right) \\
&= n2^{n+1} - 2^{n+1} + 2 \\
&= (n-1)2^{n+1} + 2.
\end{aligned}$$

□

Exercise 3. *Suppose $x \neq 1$. Prove that*

$$\sum_{j=0}^{n} jx^j = \frac{nx^{n+2} - (n+1)x^{n+1} + x}{(x-1)^2}.$$

Challenge: do this without *using mathematical induction.*

First proof. Consider first the geometric sum formula, which is valid for $x \neq 1$:

$$\sum_{j=0}^{n} x^j = \frac{1-x^{n+1}}{1-x}$$

Interpreting this as an equality of two differentiable functions, we take the derivative of both sides to obtain:

$$\sum_{j=0}^{n} jx^{j-1} = \sum_{j=1}^{n} jx^{j-1} \quad \text{(lefthand side side)}$$

and

$$\frac{-(1-x)(n+1)x^n - (1-x^{n+1})(-1)}{(1-x)^2} = \frac{nx^{n+1} - (n+1)x^n + 1}{(x-1)^2} \quad \text{(righthand side)}$$

and so

$$\sum_{j=1}^{n} jx^{j-1} \;=\; \frac{nx^{n+1}-(n+1)x^n+1}{(x-1)^2}$$

Multiplying both sides by x yields:

$$\sum_{j=1}^{n} jx^{j} \;=\; \frac{nx^{n+2}-(n+1)x^{n+1}+x}{(x-1)^2}$$

Finally, since $jx^j = 0$ when $j = 0$, we can add 0 to both sides, i.e., we can start the lefthand summation at $j = 0$:

$$\sum_{j=0}^{n} jx^{j} \;=\; \frac{nx^{n+2}-(n+1)x^{n+1}+x}{(x-1)^2}. \qquad \square$$

Second proof. For another proof, we mimic the proof of the previous exercise. Note that

$$\begin{aligned}
\sum_{j=1}^{n} jx^j &= \sum_{j=1}^{n}\sum_{k=1}^{j} x^j \\
&= \sum_{k=1}^{n}\sum_{j=k}^{n} x^j \quad \text{(swapping order of summation)} \\
&= \sum_{k=1}^{n}\left(\frac{1-x^{n+1}}{1-x} - \frac{1-x^k}{1-x}\right) \quad \text{(Geometric sum formula)} \\
&= \sum_{k=1}^{n}\left(\frac{-x^{n+1}}{1-x} + \frac{x^k}{1-x}\right) \\
&= \sum_{k=1}^{n}\frac{-x^{n+1}}{1-x} + \frac{1}{1-x}\sum_{k=1}^{n} x^k \\
&= \frac{nx^{n+1}}{x-1} + \frac{1}{1-x}\cdot\frac{1-x^{n+1}}{1-x} \quad \text{(Geometric sum formula)} \\
&= \frac{nx^{n+1}(x-1)+1-x^{n+1}}{(x-1)^2} \\
&= \frac{nx^{n+2}-(n+1)x^{n+1}+1}{(x-1)^2} \qquad \square
\end{aligned}$$

Exercise 4 (Recommended! Horner's rule)**.** *The following code fragment implements Horner's rule for evaluating a polynomial*

$$\begin{aligned}
P(x) &= \sum_{k=0}^{n} a_k x^k \\
&= a_0 + x\big(a_1 + x(a_2 + \cdots + x(a_{n-1} + xa_n))\big),
\end{aligned}$$

given coefficients $a_0, a_1, \ldots, a_n$ *and a value for* x*:*

```
1  $y = 0$
2  for $i = n$ downto 0
3      $y = a_i + x \cdot y$
```

(1) In terms of Θ*-notation, what is the running time of this code fragment for Horner's rule?*

(2) Write pseudocode to implement the naive polynomial-evaluation algorithm that computes each term of the polynomial from scratch. What is the running time of this algorithm? How does it compare to Horner's rule?

(3) Consider the following loop invariant:

At the start of each iteration of the ***for*** *loop of lines 2-3*

$$y \;=\; \sum_{k=0}^{n-(i+1)} a_{k+i+1}x^k.$$

Interpret a summation with no terms as equaling 0*. Following the structure of the loop invariant proof presented in this chapter, use this loop invariant to show that, at termination,* $y = \sum_{k=0}^{n} a_k x^k$.

(4) Conclude by arguing that the given code fragment correctly evaluates a polynomial characterized by the coefficients $a_0, a_1, \ldots, a_n$.

Proof. (1) We consider the cost and number of times each line is run:

```
1  $y = 0$                          $cost : c_1 \; times : 1$
2  for $i = n$ downto $0$           $cost : c_2 \; times : n+2$
3       $y = a_i + x \cdot y$        $cost : c_3 \; times : n+1$
```

Thus we see the running-time $T(n)$ is

$$T(n) \;=\; c_1 + c_2(n+2) + c_3(n+1) \;=\; (c_2+c_3)n + (c_1 + 2c_2 + c_3) \;=\; an + b$$

for appropriate constants $a, b > 0$. Then $T(n) = \Theta(n)$ runs in linear time.

(2) The following code will implement the naive polynomial-evaluation:

```
1   $sum = a_0$                              $cost : c_1 \; times : 1$
2   for $i = 1$ to $n$                       $cost : c_2 \; times : n+1$
3        $term = a_i$                        $cost : c_3 \; times : n$
4        for $j = 1$ to $i$                  $cost : c_4 \; times : \sum_{i=1}^{n}(i+1)$
5             $term = term \cdot x$          $cost : c_5 \; times : \sum_{i=1}^{n} i$
6        // now $term = a_i x^i$
7        $sum = sum + term$                  $cost : c_6 \; times : n$
8        // now $sum = \sum_{j=0}^{i} a_j x^j$
9   // now $sum = \sum_{j=0}^{n} a_j x^j$
10  return $sum$                             $cost : c_7 \; times : 1$
```

Thus we see that the running time $T(n)$ of this pseudocode is

$$\begin{aligned}
T(n) &= c_1 + c_2(n+1) + c_3 n + c_4\sum_{i=1}^{n}(i+1) + c_5\sum_{i=1}^{n} i + c_6 n + c_7 \\
&= c_1 + c_2(n+1) + c_3 n + c_4\frac{n(n+1)}{2} + c_4 n + c_5\frac{n(n+1)}{2} + c_6 n + c_7 \\
&= (c_4/2 + c_5/2)\,n^2 + (c_2 + c_3 + 3c_4/2 + c_5/2 + c_6)n + (c_1 + c_2 + c_7) \\
&= an^2 + bn + c
\end{aligned}$$

for appropriate constants $a, b, c > 0$. Thus $T(n) = \Theta(n^2)$ runs in quadratic time.

(3) We will use the following loop invariant (slightly rephrased):

After each time line 2 runs, the value of y is $\sum_{k=0}^{n-(i+1)} a_{k+i+1}x^k$.

(Initialization) Suppose line 2 has just run for the first time. Then $i = n$ and $y = 0$. Furthermore, $\sum_{k=0}^{n-(n+1)} a_{k+i+1}x^k$ is the empty sum (since the upper index is below the lower index), and so this summation equals $0 = y$.

(Maintenance) Suppose line 2 has just run and the current value of $i = i_0$ for some $n \geq i_0 \geq 0$. Furthermore, suppose the loop invariant is assumed true, i.e., the current value of y is $\sum_{k=0}^{n-(i_0+1)} a_{k+i_0+1}x^k$. Next in line 3 we first multiply y by x, so now

$$y \;=\; \sum_{k=0}^{n-(i_0+1)} a_{k+i_0+1}x^{k+1} \;=\; \sum_{k=1}^{n-(i_0+1)+1} a_{k+i_0}x^k$$

Next we add a_{i_0} to this new value of y obtaining:

$$y \;=\; a_{i_0} + \sum_{k=1}^{n-(i_0+1)+1} a_{k+i_0}x^k \;=\; \sum_{k=0}^{n-(i_0+1)+1} a_{k+i_0}x^k.$$

Next we go back up to line 2 and decrement the counter, so now $i = i_0 - 1$. Thus the current value of y is

$$y \;=\; \sum_{k=0}^{n-(i_0+1)+1} a_{k+i_0}x^k \;=\; \sum_{k=0}^{n-(i_0-1+1)} a_{k+(i_0-1)+1}x^k \;=\; \sum_{k=0}^{n-(i+1)} a_{k+i+1}x^k.$$

Thus the loop invariant is still true.

(Termination) Since we have shown Initialization and Maintenance, we know the loop invariant is true after the last time line 2 is run, in which case $i = -1$. The loop invariant in this case says that the current value of y is

$$y \;=\; \sum_{k=0}^{n-(-1+1)} a_{k-(-1+1)}x^k \;=\; \sum_{k=0}^{n} a_k x^k.$$

(4) Finally, since the final value of y is $y = \sum_{k=0}^{n} a_k x^k \;=\; P(x)$, we conclude that this code fragment correctly evaluates the polynomial characterized by the coefficients $a_0, \ldots, a_n$. ☐

Exercise 5. *Suppose $m, n \in \mathbb{Z}$ are such that $m > 0$. Prove that*

$$\left\lceil \frac{n}{m} \right\rceil \;=\; \left\lfloor \frac{n+m-1}{m} \right\rfloor$$

This gives us another reflection principle *between floors and ceilings when the argument is a rational number.*

Proof. By Fact 1.4.6 from the lecture notes, $n \bmod m = n - \lfloor n/m \rfloor m$ and so $n/m - \lfloor n/m \rfloor = (n \bmod m)/m$. Since $0 \leq (n \bmod m)/m < 1$, it follows that

$$\left\lceil \frac{n}{m} - \left\lfloor \frac{n}{m} \right\rfloor \right\rceil \;=\; \begin{cases} 0 & \text{if } n \bmod m = 0 \\ 1 & \text{if } n \bmod m > 0 \end{cases}$$

Next note that again by Fact 1.4.6 it follows that

$$\frac{n+m-1}{m} - \left\lfloor \frac{n}{m} \right\rfloor \;=\; \frac{(n \bmod m) + m - 1}{m}$$

Since $0 \leq n \bmod m \leq m-1$, adding $m-1$ to all sides yields $m-1 \leq (n \bmod m) + m - 1 \leq 2m-2 < 2m$. Thus:

$$\frac{(n \bmod m) + m - 1}{m} \text{ is } \begin{cases} \in [0,1) & \text{if } n \bmod m = 0 \\ \in [1,2) & \text{if } n \bmod m > 0 \end{cases}$$

Taking floors gives us

$$\left\lfloor \frac{(n \bmod m) + m - 1}{m} \right\rfloor = \begin{cases} 0 & \text{if } n \bmod m = 0 \\ 1 & \text{if } n \bmod m > 0 \end{cases}$$

Finally, since

$$\left\lceil \frac{n}{m} - \left\lfloor \frac{n}{m} \right\rfloor \right\rceil = \left\lfloor \frac{n+m-1}{m} - \left\lfloor \frac{n}{m} \right\rfloor \right\rfloor,$$

and since $\lfloor n/m \rfloor$ is an integer, it follows from Fact 1.4.1 (5),(6) that

$$\left\lceil \frac{n}{m} \right\rceil = \left\lfloor \frac{n+m-1}{m} \right\rfloor. \qquad \square$$

Exercise 6. *Find a necessary and sufficient condition on the real number $b > 1$ such that*

$$\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$$

holds for all real numbers $x \geq 1$.

Solution. We claim that the statement "for every $x \geq 1$, $\lfloor \log_b x \rfloor = \lfloor \log_b \lfloor x \rfloor \rfloor$" is true iff b is an integer ≥ 2.

($\Rightarrow$) We will show the contrapositive. Suppose $b > 1$ is *not* an integer. Then $\log_b b = 1$ whereas $\log_b \lfloor b \rfloor < 1$ since $\lfloor b \rfloor < b$ and $\log_b$ is strictly increasing when $b > 1$. Thus $\lfloor \log_b b \rfloor > \lfloor \log_b \lfloor b \rfloor \rfloor$ in this case.

($\Leftarrow$) Suppose $b \in \mathbb{Z}$ and $b \geq 2$. Let $x \geq 1$ be arbitrary, and let $m := \lfloor x \rfloor$. Thus $1 \leq m \leq x < m + 1$. Furthermore, let $k := \max\{\ell \in \mathbb{N} : b^\ell \leq m\}$. Then $\ell \geq 0$, and since $b \geq 2$, it follows that $b^{\ell+1} \geq m + 1$ (since $b^{\ell+1}$ is an integer, it has to either be $\leq m$ or $\geq m + 1$). Thus $b^\ell \leq m \leq x < m+1 \leq b^{\ell+1}$. Taking $\log_b$ of this inequality, and using that $\log_b$ is strictly increasing, we have $\ell \leq \log_b m \leq \log_b x < \log_b(m + 1) \leq \ell + 1$. Thus $\ell = \lfloor \log_b \lfloor x \rfloor \rfloor = \lfloor \log_b x \rfloor$. $\square$

Exercise 7. *Suppose $0 < \alpha < \beta$ and $0 < x$ are real numbers. Find a closed formula for the sum of all integer multiples of x in the closed interval $[\alpha, \beta]$.*

Solution. We need to know which integer multiples of x are in the range $[\alpha, \beta]$. For the lowest integer multiple $\geq \alpha$, we are looking for the least $m \in \mathbb{Z}$ such that $\alpha \leq mx$, i.e., the least $m \in \mathbb{Z}$ such that $\alpha/x \leq m$. Thus $m := \lceil \alpha/x \rceil$. Likewise, for the largest integer multiple $\leq \beta$, we are looking for the greatest $n \in \mathbb{Z}$ such that $nx \leq \beta$, i.e., the greatest $n \in \mathbb{Z}$ such that $n \leq \beta/x$. Thus $n := \lfloor \beta/x \rfloor$.

Let us first consider the case where $m \leq n$, i.e., where there actually is at least one integer multiple of x in $[\alpha, \beta]$. In this case the sum is

$$\begin{aligned} \sum_{k=\lceil \alpha/x \rceil}^{\lfloor \beta/x \rfloor} kx &= \frac{\lfloor \beta/x \rfloor (\lfloor \beta/x \rfloor + 1)}{2} - \frac{(\lceil \alpha/x \rceil - 1)\lceil \alpha/x \rceil}{2} \quad \text{(by Triangular number formula)} \\ &= \frac{1}{2}\left(\left\lfloor \frac{\beta}{x} \right\rfloor \left\lfloor \frac{\beta}{x} + 1 \right\rfloor - \left\lceil \frac{\alpha}{x} - 1 \right\rceil \left\lceil \frac{\alpha}{x} \right\rceil \right) \end{aligned}$$

Next we consider the case where $m > n$. In this case, $m = n + 1$, so $\lfloor \beta/x + 1 \rfloor = \lceil \alpha/x \rceil$ and so $\lfloor \beta/x \rfloor = \lceil \alpha/x - 1 \rceil$. Thus the above formula gives 0 which is what the value of the empty sum is. We conclude that the formula

$$\frac{1}{2}\left(\left\lfloor \frac{\beta}{x} \right\rfloor \left\lfloor \frac{\beta}{x} + 1 \right\rfloor - \left\lceil \frac{\alpha}{x} - 1 \right\rceil \left\lceil \frac{\alpha}{x} \right\rceil \right)$$

is valid in all cases. $\square$

Exercise 8. *How many of the numbers 2^m, for $0 \le m \le M$ (where $m, M \in \mathbb{N}$), have leading digit 1 when written in decimal notation? Your answer should be a closed formula.*

Solution. We start with the following claim:

Claim. *For every $n \ge 0$:*

(a) There is exactly one value of $M \ge 0$ such that $10^n \le 2^M < 10^{n+1}$ and 2^M has leading digit 1.

(b) There is exactly one value of $M \ge 0$ such that $10^n \le 2^M < 10^{n+1}$ and 2^M has leading digit $5, 6, 7, 8$, or 9.

Proof sketch of claim. We prove (a) and (b) simultaneously by induction. The base case is clear. Then for each value of $n \ge 0$, proving (a) for $n+1$ follows from the assumption that (b) holds for n, and then proving (b) for $n+1$ follows from knowing (a) is true for $n+1$, then double that number twice or three times to arrange a leading digit of $5, 6, 7, 8$ or 9. □

Now, suppose $M \ge 0$ is given. Then for $n := \lfloor \log_{10} 2^M \rfloor$ we have that $10^n \le 2^M < 10^{n+1}$, so by part (a) of the claim, there are $n+1$ values of m such that $m \le M$ and 2^m has leading digit 1 (for each $k \in \{0, \dots, n\}$, there is a unique $m \le M$ such that $10^k \le 2^m < 10^{k+1}$ and 2^k has leading digit 1, for $k = n$, this might be $m = M$ or $m = M-1$ or $m = M-2$ or $m = M-3$). Thus the total number of numbers 2^m with $0 \le m \le M$ such that 2^m has leading digit 1 is $n + 1 = \lfloor M \log_{10} 2 \rfloor + 1$. □

Exercise 9. *Suppose $x, y, z \in \mathbb{Z}$ are such that $y, z \ge 1$. Prove that $z(x \bmod y) = (zx) \bmod (zy)$.*

Proof. Take $q, r \in \mathbb{Z}$ such that $x = yq + r$ and $0 \le r < y$. Thus $x \bmod y = r$. Hence, $zx = z(yq + r)$ and $zx = (zy)q + rz$ where $0 \le rz < zy$ since $z \ge 1$. This implies $(zx) \bmod (zy) = rz = z(x \bmod y)$. □

Exercise 10. *Suppose $a, b, r, s \in \mathbb{Z}$ are such that $r, s \ge 1$. Prove that if $a \bmod rs = b \bmod rs$, then $a \bmod r = b \bmod r$ and $a \bmod s = b \bmod s$.*

Proof. Suppose $a \bmod rs = t = b \bmod rs$. Then there are $q_0, q_1 \in \mathbb{Z}$ such that $a = q_0 rs + t$ and $b = q_1 rs + t$ and $0 \le t < rs$. Next, divide t by r to get $q_3, w \in \mathbb{Z}$ such that $t = q_3 r + w$ and $0 \le w < r$. Then

$$a = q_0 rs + t = q_0 rs + q_3 r + w = r(q_0 s + q_3) + w \quad \text{with } 0 \le w < r$$

and

$$b = q_1 rs + t = q_1 rs + q_3 r + w = r(q_1 s + q_3) + w \quad \text{with } 0 \le w < r.$$

Thus $a \bmod r = w = b \bmod r$. A similar argument shows $a \bmod s = b \bmod s$. □

Exercise 11. *Suppose $b > 1$. Express $\log_b \log_b x$ in terms of $\ln \ln x$, $\ln \ln b$, and $\ln b$.*

Solution. We apply the change of base formula to the rightmost $\log_b$ first:

$$\begin{aligned} \log_b \log_b x &= \log_b (\log_b x) \\ &= \log_b \left(\frac{\ln x}{\ln b} \right) \\ &= \log_b (\ln x) - \log_b (\ln b) \\ &\qquad \text{(by the logarithmic quotient rule)} \\ &= \frac{\ln (\ln x)}{\ln b} - \frac{\ln (\ln b)}{\ln b} \\ &\qquad \text{(by applying the change of base formula to the remaining } \log_b) \\ &= \frac{\ln \ln x - \ln \ln b}{\ln b}. \end{aligned}$$

□

Exercise 12 (Programming exercise, also doable by hand)**.** *If we list all the natural numbers below 10 that are multiples of 3 or 5, we get 3, 5, 6 and 9. The sum of these multiples is 23. Find the sum of all the multiples of 3 or 5 below 1000000.*

Solution by hand. The exercise can be done by hand using the Triangular number formula several times. The first thing we do is sum up all the multiples of 3 below 1000000 (that is, ≤ 999999). We know the first multiple is $3 \cdot 1$. To know the last multiple, we divide 999999 by 3: $999999 = 333333 \cdot 3 + 0$. Thus the sum of all multiples of 3 between 0 and 999999 is

$$\sum_{k=1}^{333333} 3k = 3 \sum_{k=1}^{333333} k = \frac{3 \cdot 333333 \cdot 333334}{2}$$

Next we sum up all multiples of 5. To know the last multiple of 5 below 999999, we divide 999999 by 5: $999999 = 199999 \cdot 5 + 4$. Thus the sum of all multiples of 5 between 0 and 999999 is

$$\sum_{k=1}^{199999} 5k = 5 \sum_{k=1}^{199999} k = \frac{5 \cdot 199999 \cdot 200000}{2}$$

Finally, since some multiples of 3 are also multiples of 5 (namely, multiples of 15), we have double counted these multiples so we need to subtract the sum of these multiples once. To know the last multiple of 15, we divide 999999 by 15: $999999 = 66666 \cdot 15 + 9$. Thus the sum of all multiples of 15 between 0 and 999999 is

$$\sum_{k=1}^{66666} 15k = 15 \sum_{k=1}^{66666} k = \frac{15 \cdot 66666 \cdot 66667}{2}$$

Thus the total sum is

$$\frac{3 \cdot 333333 \cdot 333334}{2} + \frac{5 \cdot 199999 \cdot 200000}{2} - \frac{15 \cdot 66666 \cdot 66667}{2} = 233333166668 \qquad \square$$

Python solution. The following code has running time $\Theta(n)$, although since it only runs up to $i = 1000000$ it is still pretty fast. For a constant time algorithm you could implement the triangular number formula as above.

```python
my_sum = 0

for i in range(1,1000000):
    #loop will go from i=1 to i=999999
    if (i%3==0) or (i%5==0):
        my_sum+=i

print(my_sum)
```

When run, the algorithm prints:

```
233333166668
```

□

Exercise 13 (Programming exercise)**.** *Let $F_0, F_1, F_2, \ldots$ be the sequence of Fibonacci numbers. Compute the following sum:*

$$\sum_{\substack{F_n < 10^7 \\ F_n \bmod 2 = 0}} F_n$$

Proof. The following program computes successive Fibonacci numbers, remembering the previous Fibonacci number to use in later computation, and adds them to a running sum if they are even:

```python
previous_fib=0
current_fib=1
fib_sum=0
while current_fib<10**7:
    if current_fib%2==0:
        fib_sum+=current_fib #Maintains a running sum of even Fibonacci numbers
    temp=previous_fib #Store F_(n-1) in a temporary variable
    previous_fib=current_fib
    current_fib+=temp #Compute F_(n+1) from F_n and F_(n-1).

print(fib_sum)
```

When run, the algorithm prints:

```
4613732
```

□

Exercise 14 (Programming exercise)**.** *Determine the following number:*

$$\min\left\{n \in \mathbb{N} : n \geq 1 \textit{ and for each } k \in \{1, 2, \ldots, 30\}, k|n\right\}$$

Note: the above number certainly exists since the above set contains the number 30!*, so it is non-empty and thus has a minimum element by the Well-Ordering Principle.*

Python solution. First we have the following two subroutines which computes the gcd and the lcm:

```python
def Euclid(a,b):
    if b==0:
        return a
    return Euclid(b,a%b)

def LCM(a,b):
    #uses formula lcm(a,b)=a*b/gcd(a,b)
    return int(a*b/Euclid(a,b))
```

Then to compute the number in question, we run the following algorithm:

```python
lcm = 1
for k in range(1,31):
    #loop goes from k=1 to k=30
    lcm = LCM(lcm,k)

print(lcm)
```

which prints out:

```
2329089562800
```

□

Exercise 15 (Programming exercise)**.** *Let P_n denote the nth prime number. So $P_1 = 2, P_2 = 3, P_3 = 5, P_4 = 7, \ldots$. Find P_{100000}.*

Proof. The idea of the following code is that we will store a growing list of consecutive primes, which we will use to check primality of future primes. This is what the following subroutines do:

```python
import math #this allows us to use sqrt and ceil below
primes = [2,3,5] #this should always contain consecutive primes

def next_prime():
    #This will grow primes array by one more prime
    value = primes[-1]
    orig_length = len(primes)

    while len(primes) == orig_length:
        value += 1
        if is_prime(value):
            primes.append(value)


def is_prime(N):
    #assumes that array primes is big enough to detect primality of N
    #this should *only* be called by the next_prime() subroutine
    sqrtN = math.ceil(math.sqrt(N))
    for p in primes:
        if N%p == 0 :
            return False
        if p>sqrtN:
            break
            #end the for loop early if p >sqrt{N}, since N cannot have a prime
   factor greater than its sqrt
    return True
```

The following algorithm will return the nth prime, first by growing the list until it has at least n primes in it, then returning the nth prime from that list:

```
def nth_prime(n):
    #while loop grows the primes array until
    while len(primes)<n:
        next_prime()
    return primes[n-1]
    #need to shift by an index since the 1st prime is primes[0]=primes[1-1]=2
```

Now, calling

```
nth_prime(100000)
```

returns

```
1299709
```

□

Exercise 16 (Programming exercise)**.** *A unit fraction contains* 1 *in the numerator. The decimal representation of the unit fractions with denominators* 2 *to* 10 *are given:*

$$1/2 = 0.5, \quad 1/3 = 0.(3), \quad 1/4 = 0.25, \quad 1/5 = 0.2, \quad 1/6 = 0.1(6),$$

$$1/7 = 0.(142857), \quad 1/8 = 0.125, \quad 1/9 = 0.(1), \quad 1/10 = 0.1$$

where $0.1(6)$ *means* $0.166666...$, *and has a* 1*-digit recurring cycle. It can be seen that* $1/7$ *has a* 6*-digit recurring cycle. Find the value of* $d < 3000$ *for which* $1/d$ *contains the longest recurring cycle in its decimal fraction part. Hint: first analyze by hand what you would have to do to notice that* $1/7$ *has a* 6*-digit recurring cycle, think about it in terms of the Division Algorithm.*

Solution. The following subroutine compute the cycle length of d by performing long division as one would by hand and looking to see when the first repeated remainder shows up:

```python
def cycle_length(d):
    remainder_seq = [next_remainder(d,1)]
    while True:
        next_rem = next_remainder(d,remainder_seq[-1])
        for i in range(0,len(remainder_seq)):
            if next_rem == remainder_seq[i]: #checks to see if you've seen that
    remainder before, if yes then you entered a cycle
                return len(remainder_seq)-i
        remainder_seq.append(next_rem)

    return 0


def next_remainder(d,r):
    return (10*r)%d
```

The following algorithm finds the value of d which produces the longest cycle length:

```python
max_length=0
max_d=1
for n in range(1,3000):
    if cycle_length(n)>max_length:
        max_length=cycle_length(n)
        max_d=n

print(max_d)
```

When run it prints:

```
2971
```

□

Exercise 17. *Suppose $f(n)$ and $g(n)$ are asymptotically positive functions. Prove that $f(n) = \Theta(g(n))$ iff $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$.*

Proof. ($\Rightarrow$) Suppose $f(n) = \Theta(g(n))$. By definition there are $0 < c_1 \leq c_2$ and $0 < n_0$ such that for every $n \geq n_0$:

$$0 \ < \ c_1 g(n) \ \leq \ f(n) \ \leq \ c_2 g(n)$$

In particular, for this c_1 and n_0, for every $n \geq n_0$ we have

$$0 \ < \ c_1 g(n) \ \leq f(n)$$

and so $f(n) = \Omega(g(n))$. Furthermore, for this c_2 and n_0, for every $n \geq n_0$ we have

$$0 \ < \ f(n) \ \leq \ c_2 g(n)$$

and so $f(n) = O(g(n))$.

($\Leftarrow$) Suppose $f(n) = O(g(n))$ and $f(n) = \Omega(g(n))$. Then by definition there are $c_1 > 0$ and $n_1 > 0$ such that for every $n \geq n_1$:

$$0 \ < \ c_1 g(n) \ \leq \ f(n)$$

(since $f(n) = \Omega(g(n))$) and there are $c_2 > 0$ and $n_2 > 0$ such that for every $n \geq n_2$:

$$0 \ < \ f(n) \ \leq \ c_2 g(n)$$

(since $f(n) = O(g(n))$). Now if we let $n_0 := \max(n_1, n_2)$, and take the same $c_1, c_2 > 0$, then for every $n \geq n_0$ we have

$$0 \ < \ c_1 g(n) \ \leq \ f(n) \ \leq \ c_2 g(n)$$

and thus $f(n) = \Theta(g(n))$. □

Exercise 18. *Prove that for all $m \in \mathbb{N}$, $(\ln n)^m = o(n)$.*

Proof. We will prove the following claim by induction on $m \geq 0$:

$$P(m): \quad \text{“}\lim_{n\to\infty} (\ln n)^m/n = 0\text{”}$$

(Base Case) Assume $m = 0$. Then

$$\lim_{n\to\infty} \frac{(\ln n)^0}{n} \;=\; \lim_{n\to\infty} \frac{1}{n} \;=\; 0.$$

(Inductive Step) Assume for some $m \geq 0$ we know that $P(m)$ is true. We will now prove $P(m+1)$. Note that $m + 1 \geq 1$, so $\lim_{n\to\infty}(\ln n)^{m+1} = +\infty$ and $\lim_{n\to\infty} n = +\infty$. To compute the limit of the quotient, we will interpret $(\ln n)^m$ and n as functions of a continuous variable $n \in (0, +\infty)$, and employ L'Hopital's rule:

$$\lim_{n\to\infty} \frac{(\ln n)^{m+1}}{n} \;=\; \lim_{n\to\infty} \frac{(m+1)(\ln n)^m}{n} \;=\; (m+1) \lim_{n\to\infty} \frac{(\ln n)^m}{n} \;=\; 0,$$

where the last equality is true by the assumption $P(m)$.

We conclude that for every $m \geq 0$, $\lim_{n\to\infty}(\ln n)^n/n = 0$. Thus for every $m \geq 0$ we have $(\ln n)^m = o(n)$. □