```
%matplotlib inline
import matplotlib
import seaborn as sns
sns.set()
matplotlib.rcParams['figure.dpi'] = 144
```

```
%matplotlib inline
import matplotlib
import seaborn as sns
matplotlib.rcParams['savefig.dpi'] = 144
```

```
from static_grader import grader
```

Program Flow exercises

The objective of these exercises is to develop your ability to use iteration and conditional logic to build reusable functions. We will be extending our `get_primes` example from the Program Flow notebook for testing whether much larger numbers are prime. Large primes are useful for encryption. It is too slow to test every possible factor of a large number to determine if it is prime, so we will take a different approach.

Exercise 1: `mersenne_numbers`

A Mersenne number is any number that can be written as $2^p - 1$ for some p. For example, 3 is a Mersenne number $(2^2 - 1)$ as is 31 $(2^5 - 1)$. We will see later on that it is easy to test if Mersenne numbers are prime.

Write a function that accepts an exponent p and returns the corresponding Mersenne number.

```
def mersenne_number(p):
    mersnenne_num = 2**p-1
    return mersenne_num
```

Mersenne numbers can only be prime if their exponent, p, is prime. Make a list of the Mersenne numbers for all primes p between 3 and 65 (there should be 17 of them).

Hint: It may be useful to modify the `is_prime` and `get_primes` functions from the Program Flow notebook for use in this problem.

```
# we can make a list like this
my_list = [0, 1, 2]
print(my_list)
```

```
[0, 1, 2]
```

```
# we can also make an empty list and add items to it
another_list = []
print(another_list)
```

```
for item in my_list:
    another_list.append(item)

print(another_list)
```

```
[]
[0, 1, 2]
```

```
def is_prime(number):
    if number <=1:
        return False
    for factor in range(2, number):
        if number%factor ==0:
            return False

    return True

def get_primes(n_start, n_end):
    list_primes = []  # list with Mersenne prime numbers
    for p in range(n_start, n_end+1):
        if is_prime(p):
            num = 2**p-1
            list_primes.append(num)

    return list_primes
```

The next cell shows a dummy solution, a list of 17 sevens. Alter the next cell to make use of the functions you've defined above to create the appropriate list of Mersenne numbers.

```
mersennes = get_primes(3,65) #mersenne primes

print(mersennes)
```

```
[7, 31, 127, 2047, 8191, 131071, 524287, 8388607, 536870911,
2147483647, 137438953471, 2199023255551, 8796093022207,
140737488355327, 9007199254740991, 576460752303423487,
2305843009213693951]
```

```
grader.score.ip__mersenne_numbers(mersennes)
```

```
==================
Your score:  1.0
==================
```

Exercise 2: `lucas_lehmer`

We can test if a Mersenne number is prime using the Lucas-Lehmer test. First let's write a function that generates the sequence used in the test. Given a Mersenne number with exponent p, the sequence can be defined as

$$n_0 = 4$$

$$n_i = \left(n_{i-1}^2 - 2\right) mod \left(2^p - 1\right)$$

Write a function that accepts the exponent p of a Mersenne number and returns the Lucas-Lehmer sequence up to $i = p - 2$ (inclusive). Remember that the modulo operation is implemented in Python as `%`.

```
def lucas_lehmer(p):
    mer_list=[4]      # list with calculated elements, first element
is n0=4
    for i in range(1, p-1):
        val_calc=(mer_list[i-1]**2-2)%(2**p-1) # recursion with one
term
        mer_list.append(val_calc)

    return mer_list

print(lucas_lehmer(17))
```

```
[4, 14, 194, 37634, 95799, 119121, 66179, 53645, 122218, 126220,
70490, 69559, 99585, 78221, 130559, 0]
```

Use your function to calculate the Lucas-Lehmer series for $p = 17$ and pass the result to the grader.

```
ll_result = lucas_lehmer(17)

grader.score.ip__lucas_lehmer(ll_result)
```

```
==================
Your score:  1.0
==================
```

Exercise 3: `mersenne_primes`

For a given Mersenne number with exponent p, the number is prime if the Lucas-Lehmer series is 0 at position $p - 2$. Write a function that tests if a Mersenne number with exponent p is prime. Test if the Mersenne numbers with prime p between 3 and 65 (i.e. 3, 5, 7, ..., 61) are prime. Your final answer should be a list of tuples consisting of `(Mersenne exponent, 0)` (or `1`) for each Mersenne number you test, where `0` and `1` are replacements for `False` and `True` respectively.

HINT: The `zip` function is useful for combining two lists into a list of tuples

```
def ll_prime(p):
    l1_mer_exp =[]
    l2_bool=[]
    tuples_list=[]


    for i in [3, 5, 7, 11, 13, 17, 19, 23, 29, 31, 37, 43, 47, 49, 53,
59, 61]:
```

```
        test_list = lucas_lehmer(i)
        if test_list[-1]==0:
            l1_mer_exp.append(i)
            l2_bool.append(1)
        else:
            l1_mer_exp.append(i)
            l2_bool.append(0)

    tuples_list = list(zip(l1_mer_exp, l2_bool))

    return tuples_list


mersenne_primes = ll_prime(65)
grader.score.ip__mersenne_primes(mersenne_primes)
```

```
==================
Your score:  0.9411764705882354
==================
```

Exercise 4: Optimize `is_prime`

You might have noticed that the primality check `is_prime` we developed before is somewhat slow for large numbers. This is because we are doing a ton of extra work checking every possible factor of the tested number. We will use two optimizations to make a `is_prime_fast` function.

The first optimization takes advantage of the fact that two is the only even prime. Thus we can check if a number is even and as long as its greater than 2, we know that it is not prime.

Our second optimization takes advantage of the fact that when checking factors, we only need to check odd factors up to the square root of a number. Consider a number n decomposed into factors $n=a\,b$. There are two cases, either n is prime and without loss of generality, $a=n, b=1$ or n is not prime and $a, b\neq n, 1$. In this case, if $a>\sqrt{n}$, then $b<\sqrt{n}$. So we only need to check all possible values of b and we get the values of a for free! This means that even the simple method of checking factors will increase in complexity as a square root compared to the size of the number instead of linearly.

Lets write the function to do this and check the speed! `is_prime_fast` will take a number and return whether or not it is prime.

You will see the functions followed by a cell with an `assert` statement. These cells should run and produce no output, if they produce an error, then your function needs to be modified. Do not modify the assert statements, they are exactly as they should be!

```
import math
def is_prime_fast(number):
    if (number>2)and(number%2==0):
        return False
    else:
```

```
        for factor in range(3, int(math.sqrt(number))+1):
            if (factor%2!=0)and(number%factor==0):
                return False

    return True

print(is_prime_fast(125))
```

```
False
```

Run the following cell to make sure it finds the same primes as the original function.

```
for n in range(10000):
    assert is_prime(n) == is_prime_fast(n)
```

```
---------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
<ipython-input-59-e8eea54ab255> in <module>()
      1 for n in range(10000):
----> 2     assert is_prime(n) == is_prime_fast(n)

AssertionError:
```

Now lets check the timing, here we will use the `%%timeit` magic which will time the execution of a particular cell.

```
%%timeit
is_prime(67867967)
```

```
5.15 s ± 129 ms per loop (mean ± std. dev. of 7 runs, 1 loop each)
```

```
%%timeit
is_prime_fast(67867967)
```

```
713 µs ± 7.81 µs per loop (mean ± std. dev. of 7 runs, 1000 loops
each)
```

Now return a function which will find all prime numbers up to and including n. Submit this function to the grader.

```
def get_primes_fast(n):
    primes_list = []
    for i in range(2,n+1):
        if is_prime_fast(i)==True:
            primes_list.append(i)

    return primes_list

grader.score.ip__is_prime_fast(get_primes_fast)
```

```
==================
Your score:  1.0
==================
```

Exercise 5: sieve

In this problem we will develop an even faster method which is known as the Sieve of Eratosthenes (although it will be more expensive in terms of memory). The Sieve of Eratosthenes is an example of dynamic programming, where the general idea is to not redo computations we have already done (read more about it here). We will break this sieve down into several small functions.

Our submission will be a list of all prime numbers less than 2000.

The method works as follows (see here for more details)

1. Generate a list of all numbers between 0 and N; mark the numbers 0 and 1 to be not prime
2. Starting with $p=2$ (the first prime) mark all numbers of the form $n\,p$ where $n>1$ and $n\,p<¿\,N$ to be not prime (they can't be prime since they are multiples of 2!)
3. Find the smallest number greater than p which is not marked and set that equal to p, then go back to step 2. Stop if there is no unmarked number greater than p and less than $N+1$

We will break this up into a few functions, our general strategy will be to use a Python `list` as our container although we could use other data structures. The index of this list will represent numbers.

We have implemented a `sieve` function which will find all the prime numbers up to n. You will need to implement the functions which it calls. They are as follows

- `list_true` Make a list of true values of length $n+1$ where the first two values are false (this corresponds with step 1 of the algorithm above)
- `mark_false` takes a list of booleans and a number p. Mark all elements $2\,p\,,3\,p\,,\ldots n$ false (this corresponds with step 2 of the algorithm above)
- `find_next` Find the smallest `True` element in a list which is greater than some p (has index greater than p (this corresponds with step 3 of the algorithm above)
- `prime_from_list` Return indices of True values

Remember that python lists are zero indexed. We have provided assertions below to help you assess whether your functions are functioning properly.

```
def list_true(n):
    list_true=[False, False]+[True for i in range(2, n+1)]
    return list_true

print(list_true(20))
```

```
[False, False, True, True, True, True, True, True, True, True, True,
True, True, True, True, True, True, True, True, True]
```

```
assert len(list_true(20)) == 21
assert list_true(20)[0] is False
assert list_true(20)[1] is False
```

Now we want to write a function which takes a list of elements and a number p and marks elements false which are in the range $2p, 3p \ldots N$.

```
def mark_false(bool_list, p):
    bool_list[0]=False
    bool_list[1]=False
    for n in range(2, int(len(bool_list)/p)+1):
        bool_list[n*p]=False
    return bool_list

assert mark_false(list_true(6), 2) == [False, False, True, True,
False, True, False]
```

Now lets write a `find_next` function which returns the smallest element in a list which is not false and is greater than p.

```
def find_next(bool_list, p):
    for i in range(len(bool_list)):
        if (bool_list[i]!=False) and (i>p):
            smallest_elem = i
        else:
            smallest_elem = None
    return smallest_elem

assert find_next([True, True, True, True], 2) == 3
assert find_next([True, True, True, False], 2) is None
```

Now given a list of `True` and `False`, return the index of the true values.

```
def prime_from_list(bool_list):
    true_list = []
    for i in range (len(bool_list)):
        if bool_list[i]==True:
            true_list.append(i)
    return true_list

assert prime_from_list([False, False, True, True, False]) ==  [2, 3]

def sieve(n):
    bool_list = list_true(n)
    p = 2
    while p is not None:
        bool_list = mark_false(bool_list, p)
        p = find_next(bool_list, p)
    return prime_from_list(bool_list)

print(sieve(1000))
```

```
[2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69,
71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103,
105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131,
133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159,
161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187,
189, 191, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215,
217, 219, 221, 223, 225, 227, 229, 231, 233, 235, 237, 239, 241, 243,
245, 247, 249, 251, 253, 255, 257, 259, 261, 263, 265, 267, 269, 271,
273, 275, 277, 279, 281, 283, 285, 287, 289, 291, 293, 295, 297, 299,
301, 303, 305, 307, 309, 311, 313, 315, 317, 319, 321, 323, 325, 327,
329, 331, 333, 335, 337, 339, 341, 343, 345, 347, 349, 351, 353, 355,
357, 359, 361, 363, 365, 367, 369, 371, 373, 375, 377, 379, 381, 383,
385, 387, 389, 391, 393, 395, 397, 399, 401, 403, 405, 407, 409, 411,
413, 415, 417, 419, 421, 423, 425, 427, 429, 431, 433, 435, 437, 439,
441, 443, 445, 447, 449, 451, 453, 455, 457, 459, 461, 463, 465, 467,
469, 471, 473, 475, 477, 479, 481, 483, 485, 487, 489, 491, 493, 495,
497, 499, 501, 503, 505, 507, 509, 511, 513, 515, 517, 519, 521, 523,
525, 527, 529, 531, 533, 535, 537, 539, 541, 543, 545, 547, 549, 551,
553, 555, 557, 559, 561, 563, 565, 567, 569, 571, 573, 575, 577, 579,
581, 583, 585, 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607,
609, 611, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631, 633, 635,
637, 639, 641, 643, 645, 647, 649, 651, 653, 655, 657, 659, 661, 663,
665, 667, 669, 671, 673, 675, 677, 679, 681, 683, 685, 687, 689, 691,
693, 695, 697, 699, 701, 703, 705, 707, 709, 711, 713, 715, 717, 719,
721, 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 743, 745, 747,
749, 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771, 773, 775,
777, 779, 781, 783, 785, 787, 789, 791, 793, 795, 797, 799, 801, 803,
805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 825, 827, 829, 831,
833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859,
861, 863, 865, 867, 869, 871, 873, 875, 877, 879, 881, 883, 885, 887,
889, 891, 893, 895, 897, 899, 901, 903, 905, 907, 909, 911, 913, 915,
917, 919, 921, 923, 925, 927, 929, 931, 933, 935, 937, 939, 941, 943,
945, 947, 949, 951, 953, 955, 957, 959, 961, 963, 965, 967, 969, 971,
973, 975, 977, 979, 981, 983, 985, 987, 989, 991, 993, 995, 997, 999]
```

```
print(get_primes(0, 1000))
```

```
[3, 7, 31, 127, 2047, 8191, 131071, 524287, 8388607, 536870911,
2147483647, 137438953471, 2199023255551, 8796093022207,
140737488355327, 9007199254740991, 576460752303423487,
2305843009213693951, 14757395258967641292 7, 23611832414348226068 47,
94447329657392904273 91, 60446290980731458735 3087,
96714065569170333976 49407, 61897001964269013744 9562111,
15845632502852867518 7087900671, 25353012004564588029 93406410751,
10141204801825835211 973625643007, 16225927682921336339 1578010288127,
64903710731685345356 6312041152511,
10384593717069655257 060992658440191,
17014118346046923173 168730371588 4105727,
27222589353675077077 069968594541 45691647,
17422457186352049329 324779900506 5324265471,
```

696898287454081973172991196020261297061887,
713623846352979940529142984724747568191373311,
285449538541191976211657193889899027276549324 7,
182687704666362864775460604089535377456991567871,
116920130986472233456294786617302641572474603438 07,
187072209578355573530071658587684226515959365500927,
119726214130147567059245861496117904970213993920 59391,
766247770432944429179173513575154591809369561091801087,
306499108173177771671669405430061836723747824436720 4351,
313855086769334038191789471160383320805117772223201 7256447,
125542034707733615276715788464153328322047108889280 69025791,
200867255532373784442745261542645325315275374222849104412671,
803469022129495137770981046170581301261101496891396417650687,
329100911464241208430993836511470100996547173126715972669721 8047,
134799733335753198973335075435098153368185722112702862405518051 24607,
215679573337205118357336120696157045389097155380324579848828881993727,
862718293348820473429344482784628181556388621521298319395315527974911,
138034926935811275748695117245540509049022179443407731103250484475985 91,
883423532389192164791648750371459257913741948437809479060803100646309 887,
353369412955676865916659500148583703165496779375123791624321240258523 9551,
361850278866613110698659328152149712041468702080126762623304950024728 5301247,
231584178474632390847141970017375815706539969331281128078915168015826 259279871,
148213874223764730142170860811120522052185580372019921970505707530128 80593911807,
948568795032094272909893509191171341133987714380927500611236528192824 358010355711,
379427518012837709163957403676468536453595085752371000244494611277129 7432041422847,
242833611528216133864932738352939863330300854881517440156476551217363 035650651062271,
388533778445145814183892381364703781328481367810427904250362481947780 8570410416996351,
155413511378058325673556952545881512531392547124171161700144992779112 34281641667985407,
159143435651131725489722319406982668832145968255151269580948472605811 0390440106801705791,
260740604970814219042361048116400404614587954389239840081425977517360 80636970709839147486412 7,
417184967953302750467777676986240647383340727022783744130281564027777 2901915313574263597826047,
166873987181321100187111070794496258953336290809113497652112625611110 9160766125429705439130419 1,
266998379490113760299377713271194014325338065294581596243380200977777 4657225800687528702608670 71,
437450144956602384874500445423524273070633886178642487285154121281990 5

99839875184644702635404610764 7,
279968092772225526319680285071055534765205687154331191862498637620473983897520118172609686658950889471,
28668732699875893895135261191276086759957062364603514046719860492336535951106060100875231913876571081932 7,
1146749307995035755805410447651043470398282494584140561868794419693461438044242404035009276555062843277311,
18347988927920572092886567162416695526372519913346248989900710715095383008707878464560148424881005492436991,
11742712913869166139447402983946685136878412744541599353536454857661045125573042217318494991923843515159674 87,
3006134505950506531698535163890351395040873662602649434505332443561227552146698807633534717932503939880876769 27,
1923926083808324180287062504889824892826159144065695638083412763879185633373887236885462219476802521523761132339 1,
1231312693637327475383720003129487931408741852202045208373384168882678805359287831606695820465153613775207124697087,
1970100309819723960613952005007180690253986963523272333397414670212286088574860530570713312744245782040331399515340 7,
1260864198284623334792929283204595641762551656654894293374345388935863096687910739565256520156317300505812095689818111,
3227812347608635737069898965003764842912132241036529391038324195675809527521051493287056691600172289294878964965934366 71,
5164499756173817179311838344006023748659411585658447025661318713081295244033682389259290706560275662871806343945494986751,
1322111937580497197903830616065542079656809365928562438569297590548811582472622691650378420879430569695182424050046716608511,
1353842624082429130653522550851115089568572790710847937094960732721983060451965636249987502980536903367866802227247837807116287,
5415370496329716522614090203404460358274291162843391748379842930887932241807862544999950011922147613471467208908991351228465151,
5545339388241629719156828368286167406872874150751633150340959161229242615611251246079948812208279156194782421922807143657948315647,
2218135755296651887662731347314466962749149660300653260136383664491697046244500498431979524883311662477912968769122857463179326259 1,
1419606883389857208104148062281258856159455782592418086487285545274686109596480318996466895925319463985864300012238628776434768805887,
2271371013423771532966636899650014169855129252147868938379656872439497775354368510394347033480511142377382880019581806042295630089420 7,
1453677448591213781098647615776009068707282721374636120562980398361278576226795846652382101427527131121525043212532355867069203257229311,
3721414268393507279612537896386583215890643766719068468641229819804873155140597367430098179654469455671104110624082831019697160338507038 71,
5954262829429611647380060634218533145425030026750509549825967711687797048224955787888157087447151129073766576998532529631515456541611261951,
2381705131771844658952024253687413258170012010700203819930387084675118819289982315155262834978860451629506630799413011852606182616644504780 7,
3810728210834951454323238805899861213072019217120326111888619335480190110863971704248420535966176722607210609279060818964169892186631207649 2

7,
156087427515799611569079861489658315287429907133248557542957847981268586940988281006015305153174598557991346556070331144772398783964414265314508 7,
399583814440447005616844445413525287135820562261116307309972090832047582568929999375399181192126972308457847183540047730617340886948900519205142527,
639334103104715208986951112661640459417312899617786091695955345331276132110287999000638689907403155693532555493664076368987745419118240830728228044 7,
163669530394807093500659484841379957610832102302153239474164568404806689820233727744163504616295207857544334206378003550460862827294269652666426379468 7,
261871248631691349601055175746207932177331363683445183158663309447690703712373964390661607386072332572070934730204805680737380523670831444266282207150 07,
167597599124282463744675312477573076593492072757404917221544518046522050375919337210023428727086292846125398227331075635671923535149332124330420612576 0511,
686479766013060971498190079908139321726943530014330540939446345918554318339765605212255964066145455497729631139148085803712198799971664381257402829111 5057151,
274591906405224388599276031963255728690777412005732216375778538367421727335906242084902385626458182199091852455659234321484879519988665752502961131 64460228607,
719826207126911421249686161229757097419151538928306661296120891617894012907438059251046509776622537143987345701363343219713322568879087950241362428938426216821555 1,
460688772561223309599799143187044542348256984914116263229517370635452168260760357920669766257038423772151901248872539660616526440826162881544719545205927787657953 27,
471745303102692669030194322623533611364615152552055053547025787530703020299018606510765840647207345942683546878845480612471323078740599079070179281429087005456174415871,
301916993985723308179324366479061511273353697633315234270096504019649932991371908166890138014212701403317470002461107591981646770393983410604914740114615683491951626158 07,
193226876150862917234767594546599367214946366485321749932861762572575957114478021226809688329096128898123180801575108858868253933052149382787145433673354037434848490407411711,
772907504603451668939070378186397468859785465941286999731447050290303828457912084907238753316384515592492723206300435435473015732208597531148581734693416149739396162964684 7,
494660802946209068121005042039294380070262698202423679828126112185794450213063734340632802122486089979195342852032278678702730068613502419935092310203786335833213544297398271,
506532662216918085755909163048237445191949002959281848144001138878253517018177263964807989373425756138696031080481053366991595590260226478013534525648677207893210669360535830527,
324180903818827574883781864350871964922847361893940382812160728882082 2

508916334489374771131989924839287654598915078741548746211776654494592
866209641515341305165482839074293153791,
207475778444049647925620393184558057550622311612121844999782866484532 6
405706454073199853524473551897144098943305650394591197575537705887653 9
434374170569818435305909017007547618426 87,
829903113776198591702481572738232302024892464484873799991314659381305
622825816292799414097894207588576395773222601578364790302150823550615 7
737496682279273741223636068030190473707 51,
531137992816767098689588206552468627329593117727031923199444138200403 5
598608522427391625022652292856688893294862465010153465793376527072394 0
951997876658735194383127083539321903172812 7,
339928315402730943161336452193579921490939595345300430847644248448258 2
783109454353530640014497467428280891708711977606498218107760977326332 2
209278641061590524405201333465166018030600191,
543885304644369509058138323509727874385503352552480689356230797517213 2
452975126965649024023195947885249426733939164170397148972417563722131 5
534845825698544839048322133544265628848960307 1,
217554121857747803623255329403891149754201341020992275742492319006885 2
981190050786259609609278379154099770693575665668158859588967025488852 6
213938330279417935619328853417706251539584122 87,
891101683129335003640853829238338149393208692821984361441248538652202 1
810954448020519360959604241015192660760885926576778688876408936402340 3
372291400824495864296770983598924806306136567316 47,
912488123524439043728234321140058264978645701449711986115838503579855 0
334417354773011825622634742799557284619147188814621377409442750875996 5
053226394444283765039893487205299001657483844932075 51,
364995249409775617491293728456023305991458280579884794446335401431942 0
133766941909204730249053897119822913847658875525848550963771003503986
021290557777713506015957394882119600662993537972830207,
583992399055640987986069965529637289586333248927815671114136642291107 2
214027107054727568398486235391716662156254200841357681542043360560637 7
634064892444341609625531831811391361060789660756528332 7,
373755135395610232311084777938967865335253279313802029513047451066308 6
216977348515025643775031190650698663780002688538468916186907750758808 1
685801531164378630160340372359290471078905382884178132991,
239203286653190548679094257880939433814562098760833298888350368682437 5
178865503049616412016019962016447144819201720664620106359620960485637 2
278912979945202323302617838309945901490499445045874005114 87,
956813146612762194716377031523757735258248395043333195553401474729750 0
715462012198465648064079848065788579276806882658480425438483841942548 9
115651919780809293210471353239783605961997780183496020459 51,
391910664852587394955828032112131168361778542609749276898673244049305 6
293053240196491529447047105767747002071780099136913582259602981659668 0
341771026342219486499009066287015365002034290763159969980219391,
627057063764139831929324851379409869378845668175598843037877190478889 0
068885184314386447115275369228395203314848158619061731615364770655468 8
546833642147551178398414506059224584003254865221055951968351027 1,
401316520809049492434767904882822316402461227632383259544241401906488 9
644086517961207326153776236306172930121502821516199508233833453219500 0
669973530974432754174985283877903733762083113741475809259744657407,

10273702932711667006330058365000251299903007427389011444332579888806117488861485980690754953667164943802701111047223081470741078613640241920171513223929454785068796232672743355843093277117817807170494632296447,105202718030967470144819797657602573311006796056463477189965618061374643085941616442273330725551769024539659377123564354260388645003676077262556295413037616999104473422568891963833275157686454345425865034715627512693189581592767235707386820034625876761773979045465016063119822371190863000105380922197266574125287028215280054363247469065949312094107577921441162573762995177074519617763634274131844036773231242902144888720064511,275782613155099364936436410371545689780405655454255617644863469810809944371210791006433000097190429391689244677566796540832353209558436615979155575047553330706132430808858996149671100829365578879273179636604934606028770600348967705437423727721055115696583783847796289438117085048271567345759029962497646848024880749924272446637457099914453082421646959773690663827212173652660769902287067903014315801812317588193093933986970863259143372745184223339331479951185741475274045813621662589625240394934430893803101285779175998493982735923679951534365847972543945249972749854054255162024849415791137702892737463723457929162113159883256443580117751661352485851758591,289179029371721471687588745441753893207178640573601538527580357720339848228986726390361489509911551689819941427024281249599825599065947233036959036261063281298513519767830130746637524223252841238912753610632655909451254988745266864469947543547001419927068062291314858249177624616441285723525437571663787622245783832158584827037119062832388499993597209585055155728591344580177012500776216316285282091946200387572045459822604057770122494551220079820771844777304306571148032363501329423946576603711789471901808969145222512018345929631294920661032597715721502480085091455998360885653774119866519384212525315200198711376969032215538227299218443637714586638789151358605112340434124775805654747562055347407126408508313258090263755452620171577402529424076917413949640287492230608625380617615872544585318389509668184154367145724058960162017281271752812601806179444654714998039281373354488250568695072718978778398711212904759609928855585140225361330121294442200872419227452384404707852306786231942445998756897380060898818539607133650942321546909464698743315849433625922765003480450016288988711144754399686285019736718120090991211635036604543795131050361846014178702979589769250051105137720342333932222781040760521019053727537726617568176572929559009754613942621464123431600882296287828885745500823622784089099520416998111005305712631968896505259983874329375017856937076321157116496805789536226859247673436308000817682203525477342915564496652168336304859640603625881090825166872944156073823081943425974905614116740605262171928013177964545425592326671969776084891402111502344084159741989270

000285710993221138513 91,
813966605576154086191388158047028539690522256142078632860871314912602201882035164980643579207953404631705311351737456109117285358168867807661542486856790777111225290468019355281137486073219485440547487321275219968468108891293513341206527,
833501804109981784259981473840157224643094790289488520049532226470504654727204008940179025108944286342866238824179155055736100206764920635045419506541353755761894697439251819807884785738976753091120627016985825247711343504684557661395484671,
341402338963448538832888411684928399213811626102574497812288399962318706576262762061897328684623579686038011422383781910829506644690911492114603829879338498360072068071117545393309608238684878066123008826157394021462566299518794818107590521651 1,
136560935585379415533155364673971359685524650441029799124915359984927482630505104824758931473849431874415204568953512764331802657876364596845841531951735399344028827228447018157323843295473951226449203530462957608585026519807517927243036208660 47,
139838398039428521505951093426146672317977242051614514303913328624565742213637227340553145829221818239401169478608397070675765921665397347170141728718577048928285519081929746593099615534565326055883984415194068591191067156282898357496869077668331 51,
559353592157714086023804373704586689271908968206458057215653314498262968854548909362212583316887272957604677914433588282703063686661589388680566914874308195713142076327718986372398462138261304223535937660776274364764268625131593429987474631067332607,
894965747452342537638086997927338702835054349130332891545045303197220750167278254979540133307019636732167484663093741252324901898658543021888907063798893113141027322124350378195837539421218086757657500257242038983622829800210549487979962097077321727,
357986298980937015055234799170935481134021739652133156618018121278888300066911301991816053322807854692866993865237496500929960759463417208755562825519557245256410928849740151278335015768487234703063000102896815593449131920084219795191984838830928691 1,
366577970156479503416560434351037932681238261403784352376850556189581619268517173239619638602555243205495801718003196416952279817690539221765696333320266191425647911421339149090150561469309283359365121053663391676919110861662410702765924749628709797887,
600601346304376018397692615640740548904940767483960282934231951261010525009538536635792815886426510467884321534776437009534615253304179460940916872531192412803178153807272206186930267991131632985598381433432210092346427123574769369541169110979167813285969 91,
960962154087001629436308185025184878247905227974336452694771122017616840015261658617268505418282416748614914455642299215255384405286687137505466996049907860485085046091635529899088428785810612776957410293491536147754283397719630991265870577566668501257551871,
384384861634800651774523274010073951299162091189734581077908448807046736006104663446907402167312966699445965782256919686102153762114674855002186798419963144194034018436654211959635371514324245110782964117396614459101713359087852396506348231026667400503020748 7,
615015778615681042839237238416118322078659345903575329724653518091274 7

776097674615150518434677007467191135452516110714977634460193834797680034988774719410307104544294986467391354165944229187921772527425878345831345627413745405638344101571696426678408048331980 7,
100764185168393182058780629142096825889367547232841782022087232404074459563584300894626094033748090342459563254023957954193562995815789325189693256085002818471600853729058281739946654830251014910321089345590818100766759546804725978629760150674254699037463871173427 1,
161222696269429091294049006627354921422988075572546851235339571846519135301734881431401750453996944547935301206438332726709700793305262920303509209736004509554561365966493250783914647728401623856513742952945308961226815274887561565807616241078807518459942193877483 51,
644890785077716365176196026509419685691952302290187404941358287386076541206939525725607001815987778191741204825753330906838803173221051681214036838944018038218245463865973003135658590913606495426054971811781235844907261099550246263230464964315230073839768775509934 07,
103182525612434618428191364241507149710712368366429984790617325981772246593110324116097120290558044510678592772120532945094208507715368268994245894231042886114919274218555680501705374546177039268168795489884997735185161775928039402116874394290436811814363004081589452 7,
108194719976584242452959187950902601015059932372197687731806353208662815243617251220360654005792192080829316094619059953435104780186149998028910382789210025350837592882996241237756214820132135127659362899601651385169516194355519844114103684867489070385057501367856742059212 7,
173111551962534787924734700721444161624095891795516300370890165133860504389787601952577046409267507329326905751390495925496167648297839996846256612462736040561340148612793985980409943712211416204254980639362642216271225910968831750582565895787982512616092002188570787294740 47,
443165573024089057087320833846897053757685482996521728949478822742682891237856260998597238807724818763076878723559669569270189179642470391926416927904604263837030780448752604109849455903261225482892750436768364073654338332080209281491368693217235232297195525602741215474535628 7,
453801546776667194457416533859222583047869934588438250444266314488507280627564811262563572539110214413390723812925101638932673719953889681332650934174314766169119519179522666608485842844939494894482176447250804811422042452050134304247161541854448877872328218217207004645924483891 1,
116173195974826801781098632667960981260254703254640192113732176509057863840656591683216274570012214889828025296108826019566764472308195758421158639148624580139294596909957802651772375768304510692987437170496206031724042867724834381887273354714738912735316023863604993189356667876147 1,
185877113559722882849757812268737570016407525207424307381971482414492582145050546693146039312019543823724840473774121631306823155693113213473853822637799328222871355055932484242835801229287217108779899472793929650758468588359735011019637367543582260376505638181767989102970668601835 51,
118961352678222645023844999851992044810500816132751556724461748745275252572832349883613465159692508047183897903215437844036366819643592456623266446488191570062637667235796789915414912786743818949619135662588114976485419896550230407052567915227892646640963608436331513025901227905 1

```
747327,
761352657140624928152607999052749086787205223249609963036555191969761616
164661270392551261770220320515019769465805788022018327476457189917223 8
890525752442604840088107030909945545865544183516044127756246824056393 5
849506687337921474605136434657458512938502167093992521683365767858593 1
1828991,
124740019345919988228523294564802410379215703777216096343909202652325 7
432418102541115598728432897313180839029277620309527482773742745996037 9
619823739280196377000035455944285478234610759027268669891583479653399 5
249583175653445054399305553454278002759844195056679734752602647405951 8
965006204927,
199584030953471981165637271303683856606745126043545754150254724243721 1
891868964065784957965492635701089342446844192495243972437988393593660 7
391717982848314203200056729510856765175377214443629871826533567445439 2
399330810455120870388888855268448044157507120906875756041642358495230
3440099278847,
127733779810222067946007853634357668228316880667869282656163023515981 5
610796137002102373097915286848697179165980283196956142360312571899942 8
730699509022921090048036306886948329712241417243923117968981483165081 1
135573171869127735704888886737180674826080455738040048386665110943694 7
420166353846271,
817496190785421234854450263259889076661228036274363408999443350502281 9
909095276813455187826657835831661946662273812460519311106000460159634 3
876476857746694976307432364076469310158345070361107955001481492256519 1
267668299962417508511288875117956318886914916723456309674656710039646 3
4890646646161407,
209279024841067836122739267394531603625274377286237032703857497728584 1
896728390864244528083624405972905458345542095989892943643136117800866 4
032378075583153913934702685203576143400536338012443636480379262017668 8
964523084790378882178889952030196817635050218681204815276712117770149 4
653200554141732044 7,
133938575898283415118553131132500226320175601463191700930468798546293 8
813906170153116497973519619822659493341146941433531483931607115392554 4
980721968373218504918209718530288731776343256327963927347442727691308 0
937294774265842484594489569299325963286432139955971081777095755372895 6
578048354650708508671]

assert sieve(1000) == get_primes(0, 1000) #sieve(1000) returns prime
number within (0, 1000)and
                                          #get_primes(0, 1000) returns
mersenne primes; the two
                                          #list are totally different
(see above);
                                          #therefore, the assert test
between the two seems
                                          #to be wrong...

---------------------------------------------------------------------
-----
AssertionError                            Traceback (most recent call
last)
```

```
<ipython-input-69-c49169fabbae> in <module>()
----> 1 assert sieve(1000) == get_primes(0, 1000)

AssertionError:
```

```
%%timeit
sieve(1000)
```

```
236 µs ± 1.53 µs per loop (mean ± std. dev. of 7 runs, 1000 loops
each)
```

```
%%timeit
get_primes(0, 1000)
```

```
5.08 ms ± 60.8 µs per loop (mean ± std. dev. of 7 runs, 100 loops
each)
```

```
grader.score.ip__eratosthenes(sieve)
```

```
[2, 3, 5, 7, 9, 11, 13, 15, 17, 19, 21, 23, 25, 27, 29, 31, 33, 35,
37, 39, 41, 43, 45, 47, 49, 51, 53, 55, 57, 59, 61, 63, 65, 67, 69,
71, 73, 75, 77, 79, 81, 83, 85, 87, 89, 91, 93, 95, 97, 99, 101, 103,
105, 107, 109, 111, 113, 115, 117, 119, 121, 123, 125, 127, 129, 131,
133, 135, 137, 139, 141, 143, 145, 147, 149, 151, 153, 155, 157, 159,
161, 163, 165, 167, 169, 171, 173, 175, 177, 179, 181, 183, 185, 187,
189, 191, 193, 195, 197, 199, 201, 203, 205, 207, 209, 211, 213, 215,
217, 219, 221, 223, 225, 227, 229, 231, 233, 235, 237, 239, 241, 243,
245, 247, 249, 251, 253, 255, 257, 259, 261, 263, 265, 267, 269, 271,
273, 275, 277, 279, 281, 283, 285, 287, 289, 291, 293, 295, 297, 299,
301, 303, 305, 307, 309, 311, 313, 315, 317, 319, 321, 323, 325, 327,
329, 331, 333, 335, 337, 339, 341, 343, 345, 347, 349, 351, 353, 355,
357, 359, 361, 363, 365, 367, 369, 371, 373, 375, 377, 379, 381, 383,
385, 387, 389, 391, 393, 395, 397, 399, 401, 403, 405, 407, 409, 411,
413, 415, 417, 419, 421, 423, 425, 427, 429, 431, 433, 435, 437, 439,
441, 443, 445, 447, 449, 451, 453, 455, 457, 459, 461, 463, 465, 467,
469, 471, 473, 475, 477, 479, 481, 483, 485, 487, 489, 491, 493, 495,
497, 499, 501, 503, 505, 507, 509, 511, 513, 515, 517, 519, 521, 523,
525, 527, 529, 531, 533, 535, 537, 539, 541, 543, 545, 547, 549, 551,
553, 555, 557, 559, 561, 563, 565, 567, 569, 571, 573, 575, 577, 579,
581, 583, 585, 587, 589, 591, 593, 595, 597, 599, 601, 603, 605, 607,
609, 611, 613, 615, 617, 619, 621, 623, 625, 627, 629, 631, 633, 635,
637, 639, 641, 643, 645, 647, 649, 651, 653, 655, 657, 659, 661, 663,
665, 667, 669, 671, 673, 675, 677, 679, 681, 683, 685, 687, 689, 691,
693, 695, 697, 699, 701, 703, 705, 707, 709, 711, 713, 715, 717, 719,
721, 723, 725, 727, 729, 731, 733, 735, 737, 739, 741, 743, 745, 747,
749, 751, 753, 755, 757, 759, 761, 763, 765, 767, 769, 771, 773, 775,
777, 779, 781, 783, 785, 787, 789, 791, 793, 795, 797, 799, 801, 803,
805, 807, 809, 811, 813, 815, 817, 819, 821, 823, 825, 827, 829, 831,
833, 835, 837, 839, 841, 843, 845, 847, 849, 851, 853, 855, 857, 859,
861, 863, 865, 867, 869, 871, 873, 875, 877, 879, 881, 883, 885, 887,
889, 891, 893, 895, 897, 899, 901, 903, 905, 907, 909, 911, 913, 915,
```

917, 919, 921, 923, 925, 927, 929, 931, 933, 935, 937, 939, 941, 943,
945, 947, 949, 951, 953, 955, 957, 959, 961, 963, 965, 967, 969, 971,
973, 975, 977, 979, 981, 983, 985, 987, 989, 991, 993, 995, 997, 999,
1001, 1003, 1005, 1007, 1009, 1011, 1013, 1015, 1017, 1019, 1021,
1023, 1025, 1027, 1029, 1031, 1033, 1035, 1037, 1039, 1041, 1043,
1045, 1047, 1049, 1051, 1053, 1055, 1057, 1059, 1061, 1063, 1065,
1067, 1069, 1071, 1073, 1075, 1077, 1079, 1081, 1083, 1085, 1087,
1089, 1091, 1093, 1095, 1097, 1099, 1101, 1103, 1105, 1107, 1109,
1111, 1113, 1115, 1117, 1119, 1121, 1123, 1125, 1127, 1129, 1131,
1133, 1135, 1137, 1139, 1141, 1143, 1145, 1147, 1149, 1151, 1153,
1155, 1157, 1159, 1161, 1163, 1165, 1167, 1169, 1171, 1173, 1175,
1177, 1179, 1181, 1183, 1185, 1187, 1189, 1191, 1193, 1195, 1197,
1199, 1201, 1203, 1205, 1207, 1209, 1211, 1213, 1215, 1217, 1219,
1221, 1223, 1225, 1227, 1229, 1231, 1233, 1235, 1237, 1239, 1241,
1243, 1245, 1247, 1249, 1251, 1253, 1255, 1257, 1259, 1261, 1263,
1265, 1267, 1269, 1271, 1273, 1275, 1277, 1279, 1281, 1283, 1285,
1287, 1289, 1291, 1293, 1295, 1297, 1299, 1301, 1303, 1305, 1307,
1309, 1311, 1313, 1315, 1317, 1319, 1321, 1323, 1325, 1327, 1329,
1331, 1333, 1335, 1337, 1339, 1341, 1343, 1345, 1347, 1349, 1351,
1353, 1355, 1357, 1359, 1361, 1363, 1365, 1367, 1369, 1371, 1373,
1375, 1377, 1379, 1381, 1383, 1385, 1387, 1389, 1391, 1393, 1395,
1397, 1399, 1401, 1403, 1405, 1407, 1409, 1411, 1413, 1415, 1417,
1419, 1421, 1423, 1425, 1427, 1429, 1431, 1433, 1435, 1437, 1439,
1441, 1443, 1445, 1447, 1449, 1451, 1453, 1455, 1457, 1459, 1461,
1463, 1465, 1467, 1469, 1471, 1473, 1475, 1477, 1479, 1481, 1483,
1485, 1487, 1489, 1491, 1493, 1495, 1497, 1499, 1501, 1503, 1505,
1507, 1509, 1511, 1513, 1515, 1517, 1519, 1521, 1523, 1525, 1527,
1529, 1531, 1533, 1535, 1537, 1539, 1541, 1543, 1545, 1547, 1549,
1551, 1553, 1555, 1557, 1559, 1561, 1563, 1565, 1567, 1569, 1571,
1573, 1575, 1577, 1579, 1581, 1583, 1585, 1587, 1589, 1591, 1593,
1595, 1597, 1599, 1601, 1603, 1605, 1607, 1609, 1611, 1613, 1615,
1617, 1619, 1621, 1623, 1625, 1627, 1629, 1631, 1633, 1635, 1637,
1639, 1641, 1643, 1645, 1647, 1649, 1651, 1653, 1655, 1657, 1659,
1661, 1663, 1665, 1667, 1669, 1671, 1673, 1675, 1677, 1679, 1681,
1683, 1685, 1687, 1689, 1691, 1693, 1695, 1697, 1699, 1701, 1703,
1705, 1707, 1709, 1711, 1713, 1715, 1717, 1719, 1721, 1723, 1725,
1727, 1729, 1731, 1733, 1735, 1737, 1739, 1741, 1743, 1745, 1747,
1749, 1751, 1753, 1755, 1757, 1759, 1761, 1763, 1765, 1767, 1769,
1771, 1773, 1775, 1777, 1779, 1781, 1783, 1785, 1787, 1789, 1791,
1793, 1795, 1797, 1799, 1801, 1803, 1805, 1807, 1809, 1811, 1813,
1815, 1817, 1819, 1821, 1823, 1825, 1827, 1829, 1831, 1833, 1835,
1837, 1839, 1841, 1843, 1845, 1847, 1849, 1851, 1853, 1855, 1857,
1859, 1861, 1863, 1865, 1867, 1869, 1871, 1873, 1875, 1877, 1879,
1881, 1883, 1885, 1887, 1889, 1891, 1893, 1895, 1897, 1899, 1901,
1903, 1905, 1907, 1909, 1911, 1913, 1915, 1917, 1919, 1921, 1923,
1925, 1927, 1929, 1931, 1933, 1935, 1937, 1939, 1941, 1943, 1945,
1947, 1949, 1951, 1953, 1955, 1957, 1959, 1961, 1963, 1965, 1967,
1969, 1971, 1973, 1975, 1977, 1979, 1981, 1983, 1985, 1987, 1989,
1991, 1993, 1995, 1997, 1999] is too long

```
Failed validating 'maxItems' in schema:
    {'items': {'type': 'number'},
     'maxItems': 303,
     'minItems': 303,
     'type': 'array'}

On instance:
    [2,
     3,
     5,
     7,
     9,
     11,
     13,
     15,
     17,
     19,
     21,
     23,
     25,
     27,
     29,
     31,
     33,
     35,
     37,
     39,
     41,
     43,
     45,
     47,
     49,
     51,
     53,
     55,
     57,
     59,
     61,
     63,
     65,
     67,
     69,
     71,
     73,
     75,
     77,
     79,
     81,
     83,
     85,
```

87,
89,
91,
93,
95,
97,
99,
101,
103,
105,
107,
109,
111,
113,
115,
117,
119,
121,
123,
125,
127,
129,
131,
133,
135,
137,
139,
141,
143,
145,
147,
149,
151,
153,
155,
157,
159,
161,
163,
165,
167,
169,
171,
173,
175,
177,
179,
181,
183,
185,

187,
189,
191,
193,
195,
197,
199,
201,
203,
205,
207,
209,
211,
213,
215,
217,
219,
221,
223,
225,
227,
229,
231,
233,
235,
237,
239,
241,
243,
245,
247,
249,
251,
253,
255,
257,
259,
261,
263,
265,
267,
269,
271,
273,
275,
277,
279,
281,
283,
285,

287,
289,
291,
293,
295,
297,
299,
301,
303,
305,
307,
309,
311,
313,
315,
317,
319,
321,
323,
325,
327,
329,
331,
333,
335,
337,
339,
341,
343,
345,
347,
349,
351,
353,
355,
357,
359,
361,
363,
365,
367,
369,
371,
373,
375,
377,
379,
381,
383,
385,

387,
389,
391,
393,
395,
397,
399,
401,
403,
405,
407,
409,
411,
413,
415,
417,
419,
421,
423,
425,
427,
429,
431,
433,
435,
437,
439,
441,
443,
445,
447,
449,
451,
453,
455,
457,
459,
461,
463,
465,
467,
469,
471,
473,
475,
477,
479,
481,
483,
485,

487,
489,
491,
493,
495,
497,
499,
501,
503,
505,
507,
509,
511,
513,
515,
517,
519,
521,
523,
525,
527,
529,
531,
533,
535,
537,
539,
541,
543,
545,
547,
549,
551,
553,
555,
557,
559,
561,
563,
565,
567,
569,
571,
573,
575,
577,
579,
581,
583,
585,

587,
589,
591,
593,
595,
597,
599,
601,
603,
605,
607,
609,
611,
613,
615,
617,
619,
621,
623,
625,
627,
629,
631,
633,
635,
637,
639,
641,
643,
645,
647,
649,
651,
653,
655,
657,
659,
661,
663,
665,
667,
669,
671,
673,
675,
677,
679,
681,
683,
685,

687,
689,
691,
693,
695,
697,
699,
701,
703,
705,
707,
709,
711,
713,
715,
717,
719,
721,
723,
725,
727,
729,
731,
733,
735,
737,
739,
741,
743,
745,
747,
749,
751,
753,
755,
757,
759,
761,
763,
765,
767,
769,
771,
773,
775,
777,
779,
781,
783,
785,

787,
789,
791,
793,
795,
797,
799,
801,
803,
805,
807,
809,
811,
813,
815,
817,
819,
821,
823,
825,
827,
829,
831,
833,
835,
837,
839,
841,
843,
845,
847,
849,
851,
853,
855,
857,
859,
861,
863,
865,
867,
869,
871,
873,
875,
877,
879,
881,
883,
885,

887,
889,
891,
893,
895,
897,
899,
901,
903,
905,
907,
909,
911,
913,
915,
917,
919,
921,
923,
925,
927,
929,
931,
933,
935,
937,
939,
941,
943,
945,
947,
949,
951,
953,
955,
957,
959,
961,
963,
965,
967,
969,
971,
973,
975,
977,
979,
981,
983,
985,

987,
989,
991,
993,
995,
997,
999,
1001,
1003,
1005,
1007,
1009,
1011,
1013,
1015,
1017,
1019,
1021,
1023,
1025,
1027,
1029,
1031,
1033,
1035,
1037,
1039,
1041,
1043,
1045,
1047,
1049,
1051,
1053,
1055,
1057,
1059,
1061,
1063,
1065,
1067,
1069,
1071,
1073,
1075,
1077,
1079,
1081,
1083,
1085,

1087,
1089,
1091,
1093,
1095,
1097,
1099,
1101,
1103,
1105,
1107,
1109,
1111,
1113,
1115,
1117,
1119,
1121,
1123,
1125,
1127,
1129,
1131,
1133,
1135,
1137,
1139,
1141,
1143,
1145,
1147,
1149,
1151,
1153,
1155,
1157,
1159,
1161,
1163,
1165,
1167,
1169,
1171,
1173,
1175,
1177,
1179,
1181,
1183,
1185,

1187,
1189,
1191,
1193,
1195,
1197,
1199,
1201,
1203,
1205,
1207,
1209,
1211,
1213,
1215,
1217,
1219,
1221,
1223,
1225,
1227,
1229,
1231,
1233,
1235,
1237,
1239,
1241,
1243,
1245,
1247,
1249,
1251,
1253,
1255,
1257,
1259,
1261,
1263,
1265,
1267,
1269,
1271,
1273,
1275,
1277,
1279,
1281,
1283,
1285,

1287,
1289,
1291,
1293,
1295,
1297,
1299,
1301,
1303,
1305,
1307,
1309,
1311,
1313,
1315,
1317,
1319,
1321,
1323,
1325,
1327,
1329,
1331,
1333,
1335,
1337,
1339,
1341,
1343,
1345,
1347,
1349,
1351,
1353,
1355,
1357,
1359,
1361,
1363,
1365,
1367,
1369,
1371,
1373,
1375,
1377,
1379,
1381,
1383,
1385,

1387,
1389,
1391,
1393,
1395,
1397,
1399,
1401,
1403,
1405,
1407,
1409,
1411,
1413,
1415,
1417,
1419,
1421,
1423,
1425,
1427,
1429,
1431,
1433,
1435,
1437,
1439,
1441,
1443,
1445,
1447,
1449,
1451,
1453,
1455,
1457,
1459,
1461,
1463,
1465,
1467,
1469,
1471,
1473,
1475,
1477,
1479,
1481,
1483,
1485,

1487,
1489,
1491,
1493,
1495,
1497,
1499,
1501,
1503,
1505,
1507,
1509,
1511,
1513,
1515,
1517,
1519,
1521,
1523,
1525,
1527,
1529,
1531,
1533,
1535,
1537,
1539,
1541,
1543,
1545,
1547,
1549,
1551,
1553,
1555,
1557,
1559,
1561,
1563,
1565,
1567,
1569,
1571,
1573,
1575,
1577,
1579,
1581,
1583,
1585,

1587,
1589,
1591,
1593,
1595,
1597,
1599,
1601,
1603,
1605,
1607,
1609,
1611,
1613,
1615,
1617,
1619,
1621,
1623,
1625,
1627,
1629,
1631,
1633,
1635,
1637,
1639,
1641,
1643,
1645,
1647,
1649,
1651,
1653,
1655,
1657,
1659,
1661,
1663,
1665,
1667,
1669,
1671,
1673,
1675,
1677,
1679,
1681,
1683,
1685,

1687,
1689,
1691,
1693,
1695,
1697,
1699,
1701,
1703,
1705,
1707,
1709,
1711,
1713,
1715,
1717,
1719,
1721,
1723,
1725,
1727,
1729,
1731,
1733,
1735,
1737,
1739,
1741,
1743,
1745,
1747,
1749,
1751,
1753,
1755,
1757,
1759,
1761,
1763,
1765,
1767,
1769,
1771,
1773,
1775,
1777,
1779,
1781,
1783,
1785,

1787,
1789,
1791,
1793,
1795,
1797,
1799,
1801,
1803,
1805,
1807,
1809,
1811,
1813,
1815,
1817,
1819,
1821,
1823,
1825,
1827,
1829,
1831,
1833,
1835,
1837,
1839,
1841,
1843,
1845,
1847,
1849,
1851,
1853,
1855,
1857,
1859,
1861,
1863,
1865,
1867,
1869,
1871,
1873,
1875,
1877,
1879,
1881,
1883,
1885,

1887,
1889,
1891,
1893,
1895,
1897,
1899,
1901,
1903,
1905,
1907,
1909,
1911,
1913,
1915,
1917,
1919,
1921,
1923,
1925,
1927,
1929,
1931,
1933,
1935,
1937,
1939,
1941,
1943,
1945,
1947,
1949,
1951,
1953,
1955,
1957,
1959,
1961,
1963,
1965,
1967,
1969,
1971,
1973,
1975,
1977,
1979,
1981,
1983,
1985,

```
    1987,
    1989,
    1991,
    1993,
    1995,
    1997,
    1999]
```