1.How many seconds are in an hour? Use the interactive interpreter as a calculator and
multiply the
number of seconds in a minute (60) by the number of minutes in an hour (also 60).
sol. 60
2. Assign the result from the previous task (seconds in an hour) to a variable called
seconds_per_hour.
3. How many seconds do you think there are in a day? Make use of the variables seconds
per hour
and minutes per hour.
4. Calculate seconds per day again, but this time save the result in a variable called
seconds_per_day
5. Divide seconds_per_day by seconds_per_hour. Use floating-point (/) division.
6. Divide seconds_per_day by seconds_per_hour, using integer (//) division. Did this number
agree
with the floating-point value from the previous question, aside from the final .0?
7. Write a generator, genPrimes, that returns the sequence of prime numbers on successive
calls to
its next() method: 2, 3, 5, 7, 11, …

2. To assign the result from the previous task to a variable called `seconds_per_hour`, we can use the following code:

makefile

```
seconds_per_hour = 60 * 60
```

3. There are 24 hours in a day, so to calculate the number of seconds in a day, we can multiply the number of seconds in an hour by 24. We can use the `seconds_per_hour` variable we defined earlier to simplify this calculation:

makefile

```
seconds_per_day = seconds_per_hour * 24
```

4. We can calculate `seconds_per_day` and save the result in a variable called `seconds_per_day` using the same code as in step 3:

makefile

```
seconds_per_day = seconds_per_hour * 24
```

5. To calculate the ratio of seconds per day to seconds per hour using floating-point division, we can divide `seconds_per_day` by `seconds_per_hour` using the `/` operator:

```
seconds_per_day / seconds_per_hour
```

The result is `24.0`, which means there are 24 times as many seconds in a day as there are in an hour.

6. To calculate the ratio of seconds per day to seconds per hour using integer division, we can use the `//` operator:

arduino

```
seconds_per_day // seconds_per_hour
```

The result is `24`, which is the same as the floating-point value from the previous question, except that it doesn't include the `.0`.

7. Here is an implementation of the `genPrimes` generator using the Sieve of Eratosthenes algorithm:

python

```
def genPrimes():
    primes = []
    n = 2
    while True:
        if all(n % p != 0 for p in primes):
            yield n
            primes.append(n)
        n += 1
```

This generator keeps track of the prime numbers it has found so far in a list called `primes`. It starts with `n = 2`, which is the first prime number, and then checks if `n` is divisible by any of the prime numbers in `primes`. If it isn't, then `n` must be a new prime number, so the generator yields it and adds it to the list of primes. The generator then increments `n` and repeats the process. This way, the generator produces an infinite sequence of prime numbers on successive calls to its `next()` method.