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).

Ans: To calculate the number of seconds in an hour, you can use the interactive interpreter as a calculator. Here's the calculation:

```

>>> seconds\_in\_minute = 60

>>> minutes\_in\_hour = 60

>>> seconds\_in\_hour = seconds\_in\_minute \* minutes\_in\_hour

>>> seconds\_in\_hour

3600

Therefore, there are 3600 seconds in an hour.

2. Assign the result from the previous task (seconds in an hour) to a variable called seconds\_per\_hour.

Ans : Certainly! Here's how you can assign the result from the previous task (3600 seconds) to a variable called `seconds\_per\_hour`:

```

>>> seconds\_per\_hour = 3600

>>> seconds\_per\_hour

3600

```

Now, the variable `seconds\_per\_hour` holds the value 3600, which represents the number of seconds in an 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.

Ans : To calculate the number of seconds in a day, you can make use of the variables `seconds\_per\_hour` and `minutes\_in\_hour`. Here's the calculation:

```python

>>> minutes\_in\_hour = 60

>>> seconds\_per\_hour = 60 \* minutes\_in\_hour

>>> hours\_in\_day = 24

>>> seconds\_per\_day = seconds\_per\_hour \* hours\_in\_day

>>> seconds\_per\_day

86400

```

Therefore, there are 86,400 seconds in a day.

4. Calculate seconds per day again, but this time save the result in a variable called seconds\_per\_day

Ans: Apologies for the confusion. Let's calculate the number of seconds in a day again and save the result in a variable called `seconds\_per\_day`. Here's the updated calculation:

```python

>>> seconds\_per\_hour = 3600

>>> hours\_in\_day = 24

>>> seconds\_per\_day = seconds\_per\_hour \* hours\_in\_day

>>> seconds\_per\_day

86400

```

So, the variable `seconds\_per\_day` now holds the value 86,400, which represents the number of seconds in a day.

5. Divide seconds\_per\_day by seconds\_per\_hour. Use floating-point (/) division.

Ans: Certainly! To divide `seconds\_per\_day` by `seconds\_per\_hour` using floating-point division, you can perform the following calculation:

```python

>>> seconds\_per\_day = 86400

>>> seconds\_per\_hour = 3600

>>> division\_result = seconds\_per\_day / seconds\_per\_hour

>>> division\_result

24.0

```

The result of dividing `seconds\_per\_day` by `seconds\_per\_hour` is 24.0, indicating that there are 24 hours in a day.

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?

Ans: Certainly! To divide `seconds\_per\_day` by `seconds\_per\_hour` using integer division, you can use the double forward slash (`//`) operator. Here's the calculation:

```python

>>> seconds\_per\_day = 86400

>>> seconds\_per\_hour = 3600

>>> division\_result = seconds\_per\_day // seconds\_per\_hour

>>> division\_result

24

```

The result of dividing `seconds\_per\_day` by `seconds\_per\_hour` using integer division is 24. This number agrees with the floating-point value from the previous question, aside from the final `.0`. The only difference is that integer division discards the decimal part of the result, whereas floating-point division includes it.

7. Write a generator, genPrimes, that returns the sequence of prime numbers on successive calls to its next() method: 2, 3, 5, 7, 11, ...

Ans: Certainly! Here's an implementation of the `genPrimes` generator that returns the sequence of prime numbers on successive calls to its `next()` method:

```python

def genPrimes():

primes = []

num = 2

while True:

if all(num % prime != 0 for prime in primes):

primes.append(num)

yield num

num += 1

```

The `genPrimes` function initializes an empty list `primes` to keep track of prime numbers found so far. It starts with the number 2 and enters an infinite loop. Within each iteration, it checks if the current number `num` is divisible by any of the prime numbers in `primes`. If it is not divisible by any of them, it is considered a prime number, added to `primes`, and yielded using the `yield` keyword. Finally, `num` is incremented by 1 to move on to the next number.

Here's an example of how you can use the `genPrimes` generator to obtain prime numbers:

```python

primes\_generator = genPrimes()

print(next(primes\_generator)) # Output: 2

print(next(primes\_generator)) # Output: 3

print(next(primes\_generator)) # Output: 5

print(next(primes\_generator)) # Output: 7

print(next(primes\_generator)) # Output: 11

```

Each call to `next(primes\_generator)` will produce the next prime number in the sequence.