**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**

To find the number of seconds in an hour, we can multiply the number of seconds in a minute (60) by the number of minutes in an hour (60):

```

60 seconds/minute \* 60 minutes/hour = 3600 seconds/hour

```

So there are 3600 seconds in an hour.

To compute this in Python interactive interpreter, we can simply enter the calculation:

```

>>> 60 \* 60

3600

```

So the output will be `3600`, which is the number of seconds in an hour.

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

To assign the result from the previous task (seconds in an hour) to a variable called `seconds\_per\_hour`, we can do the following:

```

seconds\_per\_hour = 60 \* 60

```

This will assign the value `3600` to the variable `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.**

To calculate the number of seconds in a day, we can use the `seconds\_per\_hour` variable that we computed earlier, along with the number of hours in a day (24):

```

seconds\_per\_day = seconds\_per\_hour \* 24

```

We can also compute this using the number of seconds per minute, which is 60, and the number of minutes per hour, which is also 60, as follows:

```

seconds\_per\_day = 60 \* 60 \* 24

```

Either way, we get the same result:

```

>>> seconds\_per\_day = seconds\_per\_hour \* 24

>>> seconds\_per\_day

86400

>>> seconds\_per\_day = 60 \* 60 \* 24

>>> 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**

To calculate the number of seconds in a day, we can multiply the number of seconds in an hour (3600) by the number of hours in a day (24). We can do this using the previously calculated variable `seconds\_per\_hour` and saving the result in a new variable `seconds\_per\_day`.

Here's how to do it in Python:

```python

seconds\_per\_hour = 60 \* 60

seconds\_per\_day = seconds\_per\_hour \* 24

print(seconds\_per\_day)

```

This will output:

```

86400

```

So there are 86,400 seconds in a day.

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

Sure! We can divide `seconds\_per\_day` by `seconds\_per\_hour` to find out how many hours are in a day:

```python

seconds\_per\_hour = 60 \* 60

seconds\_per\_day = seconds\_per\_hour \* 24

hours\_per\_day = seconds\_per\_day / seconds\_per\_hour

print(hours\_per\_day)

```

This will output:

```

24.0

```

So 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?**

Yes, the integer division result agrees with the floating-point division result from the previous question, except for the decimal part. Here's the calculation:

```

seconds\_per\_day // seconds\_per\_hour

```

Output:

```

24

```

This means there are 24 hours in a day. The result is the same as the previous calculation using floating-point division, except for the decimal part.

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

Here's an implementation of the `genPrimes` generator function in Python that yields prime numbers on successive calls to its `next()` method:

```python

def genPrimes():

primes = []

num = 2

while True:

is\_prime = True

for p in primes:

if num % p == 0:

is\_prime = False

break

if is\_prime:

primes.append(num)

yield num

num += 1

```

This implementation keeps a list of all the prime numbers found so far in the `primes` variable. It starts with `num` set to 2, which is the first prime number. It then enters an infinite loop and checks if the current value of `num` is prime by checking if it is divisible by any of the previously found prime numbers in the `primes` list. If it is prime, it appends it to the `primes` list and yields it. Finally, it increments `num` by 1 and repeats the process with the next value of `num`.

Here's an example usage of the `genPrimes` function to print the first 10 prime numbers:

```python

primes = genPrimes()

for i in range(10):

print(next(primes))

```

Output:

```

2

3

5

7

11

13

17

19

23

29

```