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 by multiplying the number of seconds in a minute (60) by the number of minutes in an hour (60):

# Using the interactive interpreter as a calculator

seconds\_in\_minute = 60

minutes\_in\_hour = 60

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

print(seconds\_in\_hour)

When you run this code, it will output:

3600

So, 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: To assign the result from the previous task (3600 seconds in an hour) to a variable called `seconds\_per\_hour`, you can do the following:

seconds\_per\_hour = 3600

print(seconds\_per\_hour)

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 previously defined variable `seconds\_per\_hour` (3600 seconds) and the `minutes\_in\_hour` variable (60 minutes). There are 24 hours in a day.

seconds\_per\_hour = 3600

minutes\_in\_hour = 60

hours\_in\_day = 24

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

print(seconds\_per\_day)

When you run this code, it will output:

86400

So, 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: Let's calculate the number of seconds in a day and save the result in a variable called `seconds\_per\_day`:

seconds\_per\_hour = 3600

minutes\_in\_hour = 60

hours\_in\_day = 24

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

print(seconds\_per\_day)

When you run this code, it will output:

86400

Now, the variable `seconds\_per\_day` 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: To divide `seconds\_per\_day` by `seconds\_per\_hour` using floating-point division, you can simply use the `/` operator. Here's how you can do it:

seconds\_per\_day = 86400

seconds\_per\_hour = 3600

result = seconds\_per\_day / seconds\_per\_hour

print(result)

When you run this code, it will output:

24.0

So, `seconds\_per\_day` divided by `seconds\_per\_hour` gives you 24.0, indicating that there are 24 hours in a day. The result is a floating-point number because we used 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?

ANS: To perform integer division (using `//`) between `seconds\_per\_day` and `seconds\_per\_hour`, you can use the following code:

seconds\_per\_day = 86400

seconds\_per\_hour = 3600

result\_integer\_division = seconds\_per\_day // seconds\_per\_hour

print(result\_integer\_division)

When you run this code, it will output:

24

The result of integer division is 24, which agrees with the floating-point value from the previous question (24.0). The only difference is that the floating-point value includes the decimal `.0` because it represents a floating-point number, while the integer division result doesn't include the decimal part since it represents a whole number. Other than that, both values are the same, indicating that there are 24 hours in a day.

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: Sure, here's a Python generator called `genPrimes` that yields the sequence of prime numbers on successive calls to its `next()` method:

def is\_prime(num):

if num < 2:

return False

for i in range(2, int(num\*\*0.5) + 1):

if num % i == 0:

return False

return True

def genPrimes():

num = 2

while True:

if is\_prime(num):

yield num

num += 1

# Usage

prime\_generator = genPrimes()

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

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

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

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

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

# and so on...

The `genPrimes` generator uses the `is\_prime` function to check if a number is prime or not. The `is\_prime` function returns `True` if the number is prime and `False` otherwise. The `genPrimes` generator starts from 2 and continuously yields the next prime number on each call to `next()`.