**FSDS MAY BATCH 2022(Python Basics 15)**

**Submitted by: Shubham Tiwari**

Q1.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: In hour there are 60\*60=3600seconds

To calculate the number of seconds in an hour, we can use the interactive interpreter in Python as a calculator and multiply the number of seconds in a minute (60) by the number of minutes in an hour (also 60). Here's the calculation:

**60 \* 60**

**3600**

Q2. Assign the result from the previous task (seconds in an hour) to a variable called seconds\_per\_hour.

Ans: Yes, we can assign the result of the previous calculation (3600 seconds) to a variable called **seconds\_per\_hour** in Python.

seconds\_per\_hour = 60 \* 60

This assigns the value 3600 to the variable **seconds\_per\_hour**. Now, we can use this variable to refer to the number of seconds in an hour throughout our code.

Q3. How many seconds do you think there are in a day? Make use of the variables seconds per hour and minutes per hour.

Ans: There are 86,400 seconds in a day, since there are 60 seconds in a minute, 60 minutes in an hour, and 24 hours in a day. Alternatively, you could calculate it as:

1 day = 24 hours 1 hour = 60 minutes 1 minute = 60 seconds

So, 1 day = 24 hours/day x 60 minutes/hour x 60 seconds/minute = 86,400 seconds/day.

Q4. Calculate seconds per day again, but this time save the result in a variable called seconds\_per\_day.

Ans: We can save the result in a variable called **seconds\_per\_day** in Python:

**# Define variables for seconds, minutes, and hours**

seconds\_per\_minute = 60

minutes\_per\_hour = 60

seconds\_per\_hour = seconds\_per\_minute \* minutes\_per\_hour

**# Calculate seconds per day**

hours\_per\_day = 24

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

**# Print the result**

print(seconds\_per\_day)

This will output **86400**, which is the number of seconds in a day. The value of **86400** will be stored in the variable **seconds\_per\_day**.

Q5. Divide seconds\_per\_day by seconds\_per\_hour. Use floating-point (/) division.

Ans: We can divide **seconds\_per\_day** by **seconds\_per\_hour** using floating-point division in Python:

**# Define variables for seconds, minutes, and hours**

seconds\_per\_minute = 60

minutes\_per\_hour = 60

seconds\_per\_hour = seconds\_per\_minute \* minutes\_per\_hour

**# Calculate seconds per day**

hours\_per\_day = 24

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

**# Divide seconds\_per\_day by seconds\_per\_hour**

seconds\_per\_hour\_float = seconds\_per\_day / seconds\_per\_hour

**# Print the result**

print(seconds\_per\_hour\_float)

This will output **24.0**, which is the result of dividing **seconds\_per\_day** by **seconds\_per\_hour** using floating-point division. The value **24.0** will be stored in the variable **seconds\_per\_hour\_float**.

Q6. 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: Assume **seconds\_per\_day** and **seconds\_per\_hour** are defined as follows:

seconds\_per\_day = 86400

seconds\_per\_hour = 3600

Then the result of the integer division **seconds\_per\_day // seconds\_per\_hour** is:

**24**

This agrees with the floating-point value from the previous question, which was 24.0. The only difference is that the floating-point value includes a decimal point and a zero after it, while the integer value does not.

Q7. Write a generator, genPrimes, that returns the sequence of prime numbers on successive calls to its next() method: 2, 3, 5, 7, 11, ...

Ans: Here's an implementation of the genPrimes generator in Python that uses a basic trial division algorithm to generate the sequence of prime numbers:

def genPrimes():

"""Generate an infinite sequence of prime numbers."""

primes = [2] **# Start with the first prime number**

yield 2 **# Yield the first prime number**

n = 3 **# Start testing for primes from 3**

while True:

is\_prime = True

for prime in primes:

if n % prime == 0:

is\_prime = False

break

if is\_prime:

primes.append(n)

yield n

n += 2 # Only test odd numbers for primality

This generator maintains a list of all the prime numbers generated so far, and checks each new candidate number for primality by dividing it by each of the previously generated primes. If none of the primes divide the candidate number evenly, then it is a new prime and is added to the list of primes and yielded by the generator. The generator starts with the first prime number 2, and then only tests odd numbers for primality, since even numbers are never prime (except for 2). The **yield** statement is used to return each prime number in the sequence one at a time, so that the generator can be used in a **for** loop or other iterable context.