**1. To obtain the sum of the first n natural numbers for some given n without using any formula,**

**Source code:**

n = int(input("Enter n'th natural term: "))

i = 1

sum = 0

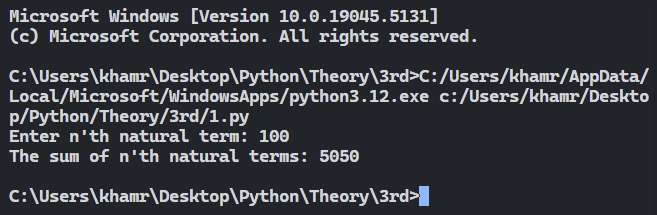
while i <= n:

    sum = sum + i

    i = i + 1

print("The sum of n'th natural terms:",sum)

**Output:**

****

**2. To obtain all the whole numbers which are divisible by some given p but not by some other given q.**

**Source code:**

p = int(input("Enter value of p: "))

q = int(input("Enter value of q: "))

limit = int(input("Enter value of upper limit: "))

i = 1

print("Whle numbers which are divided by p and not by q in the given range are:")

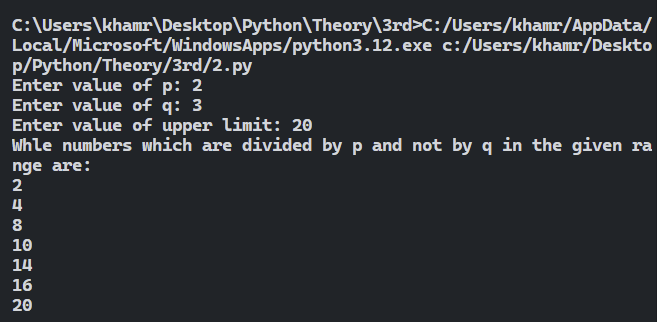
while i <= limit:

if i % p == 0 and i % q != 0:

print(i)

i = i + 1

**Output:**

****

**3. To generate a table of any given number n.**

**Source code:**

n = int(input("Enter the number to generate table:"))

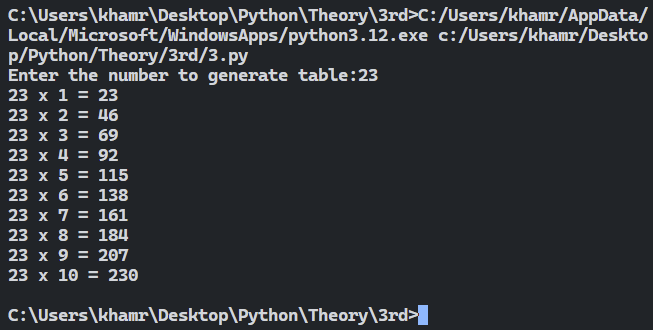
i = 1

while i <= 10:

print(n, "x", i, "=", n \* i)

i = i + 1

**Output:**

****

**4. To obtain all the groups of three successive numbers within 1000 that have the property that the square of the middle one is greater by unity than the product of the other two numbers. For example, 182 =17\*19+1.**

**Source code:**

**Output:**

**5. To find out the series of five consecutive numbers within 1000, for which the sum of the squares of the first three is equal to the sum of the squares of the last two. For example, (−2)2+(−1)2+02 = 12+22**

**Source code:**

i = 1

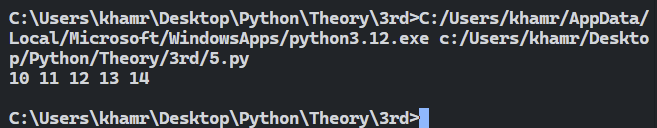
while i <= 1000 - 4:

    if i \*\* 2 + (i + 1) \*\* 2 + (i + 2) \*\* 2 == (i + 3) \*\* 2 + (i + 4) \*\* 2:

        print(i, i + 1, i + 2, i + 3, i + 4)

    i = i + 1

**Output:**

****

**6. To find out Pythagorean numbers x, y, z within 100 such that z 2 = x 2 + y 2.**

**Source code:**

x = 1

while x <= 100:

    y = x

    while y <= 100:

        z = y

        while z <= 100:

            if z \* z == x \* x + y \* y:

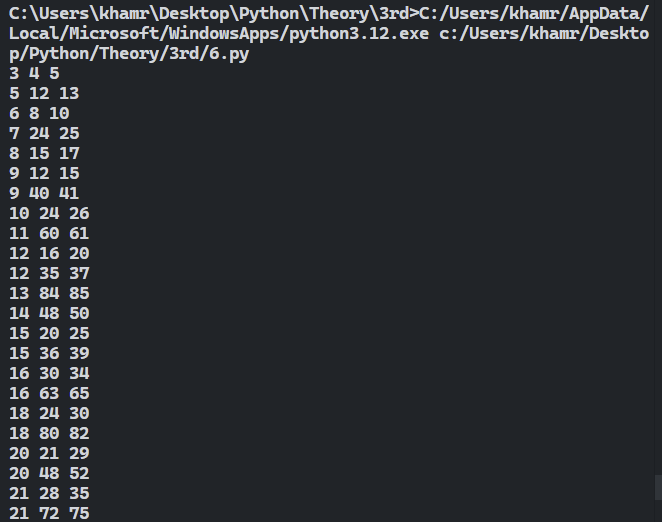
                print(x, y, z)

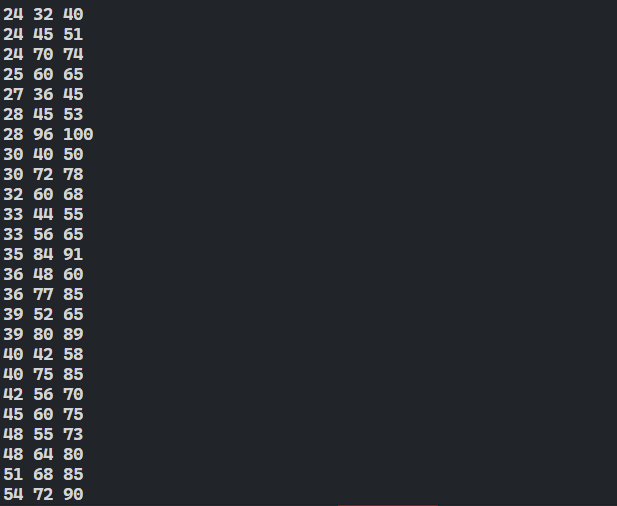
            z = z + 1

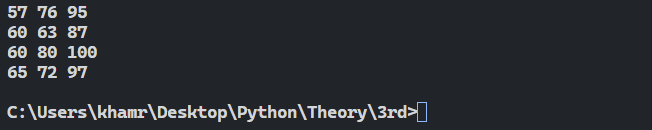
        y = y + 1

    x = x + 1

**Output:**

****

****

****

**7.** **To print the value of a given amount of currency value in words.**

**Source code:**

**Output:**

**8 a) To obtain the sum of the following series for some given values of x and n:**

****

**Source code:**

x = float(input("Enter the value of x: "))

n = int(input("Enter the number of terms n: "))

total\_a = 0

sign = 1

for i in range(n):

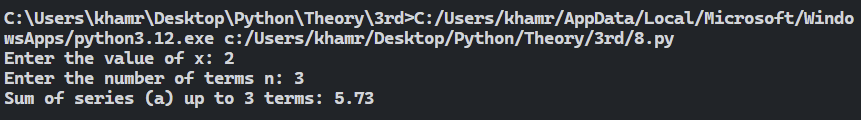
    term = sign \* x\*\*(2\*i + 1) / (2\*i + 1)

    total\_a += term

    sign \*= -1

print(f"Sum of series (a) up to {n} terms: {total\_a:.2f}")

**Output:**

****

**8 b) *x*2*/*2 − *x*4*/*4 + *x*6*/*6 ±………….. up to n terms.**

**Source code:**

x = float(input("Enter the value of x: "))

n = int(input("Enter the number of terms n: "))

total\_b = 0

sign = 1

for i in range(n):

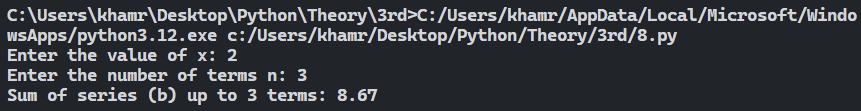
    term = sign \* x\*\*(2\*i + 2) / (2\*i + 2)

    total\_b += term

    sign \*= -1

print(f"Sum of series (b) up to {n} terms: {total\_b:.2f}")

**Output:**

****

**8 c) *x*1*/*1! − *x*3*/*3! + *x*5*/*5! ± …………. up to n terms.**

**Source code:**

x = float(input("Enter the value of x: "))

n = int(input("Enter the number of terms n: "))

total\_c = 0

sign = 1

for i in range(n):

    factorial\_c = 1

    for j in range(1, 2\*i + 2):

        factorial\_c \*= j

    term = sign \* x\*\*(2\*i + 1) / factorial\_c

    total\_c += term

    sign \*= -1

print(f"Sum of series (c) up to {n} terms: {total\_c:.2f}")

**Output:**

****

**8 d) *x*2*/*2! − *x*4*/*4! + *x*6*/*6! ±………….. up to n terms.**

**Source code:**

x = float(input("Enter the value of x: "))

n = int(input("Enter the number of terms n: "))

total\_d = 0

sign = 1

for i in range(n):

    factorial\_d = 1

    for j in range(1, 2\*i + 3):

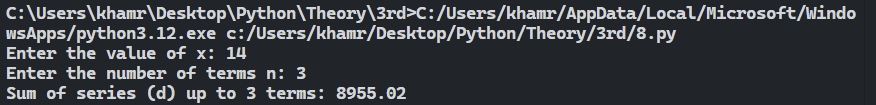
        factorial\_d \*= j

    term = sign \* x\*\*(2\*i + 2) / factorial\_d

    total\_d += term

    sign \*= -1

print(f"Sum of series (d) up to {n} terms: {total\_d:.2f}")

**Output:**

****

**8 e)**

**Source code:**

x = float(input("Enter the value of x: "))

n = int(input("Enter the number of terms (n): "))

total\_sum = 0

sign = 1

numerator = 1

denominator = 1

for i in range(n):

    exponent = 2 \* i + 1

    if i > 0:

        numerator \*= (2 \* i - 1)

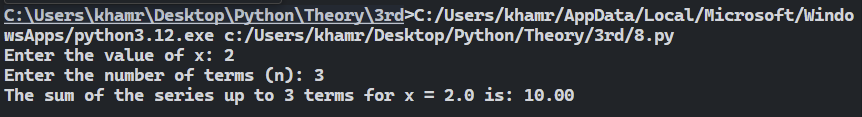
    denominator \*= (2 \* i) if i > 0 else 1

    term = sign \* numerator \* x\*\*exponent / denominator

    total\_sum += term

    sign \*= -1

print(f"The sum of the series up to {n} terms for x = {x} is: {total\_sum:.2f}")

**Output:**

**9.** **Twin prime numbers are primes that differ by 2(e.g. 3 and 5, 101 and 103). To find out all twin primes within 10000.**

**Source code:**

for num in range(2, 10000):

    is\_prime = True

    if num < 2:

        is\_prime = False

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

        if num % i == 0:

            is\_prime = False

            break

    is\_twin\_prime = True

    twin\_num = num + 2

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

        if twin\_num % i == 0:

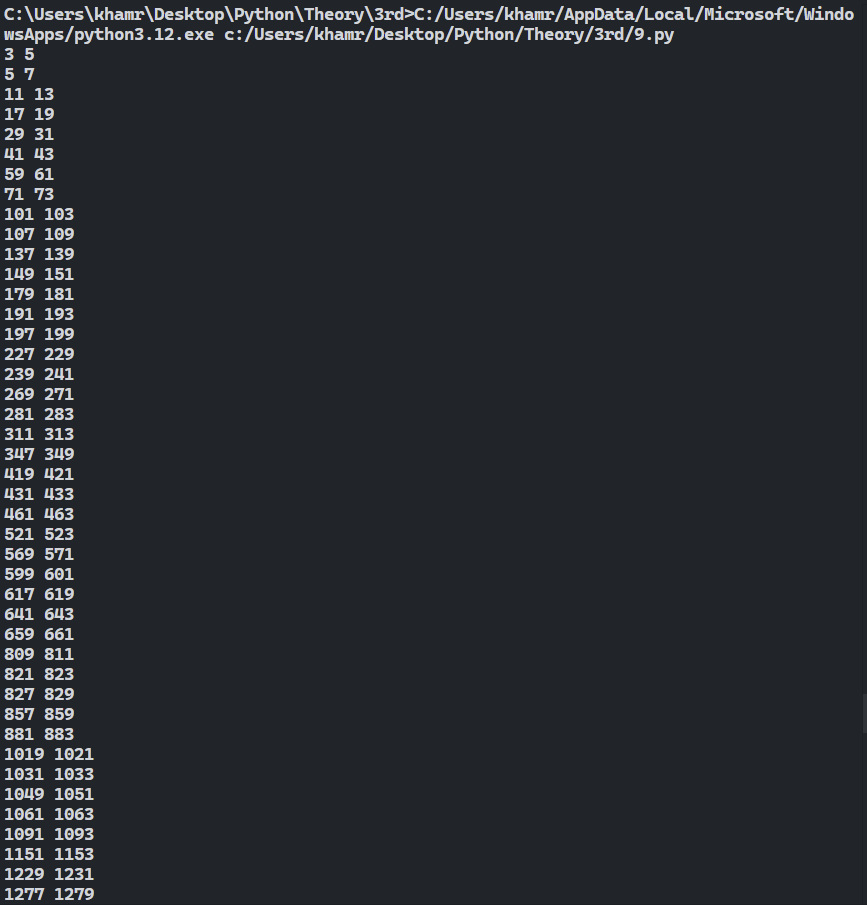
            is\_twin\_prime = False

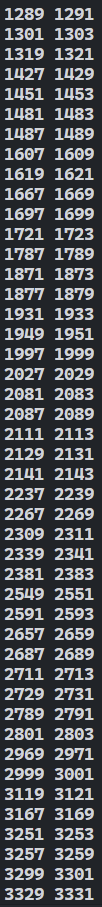
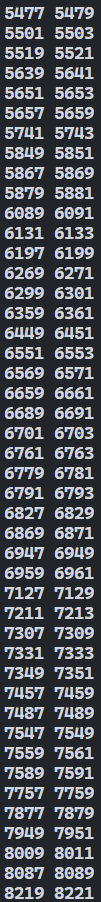
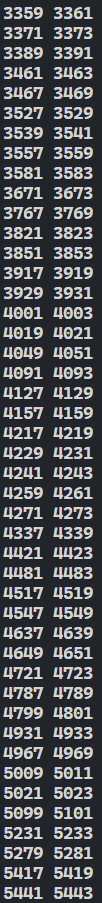
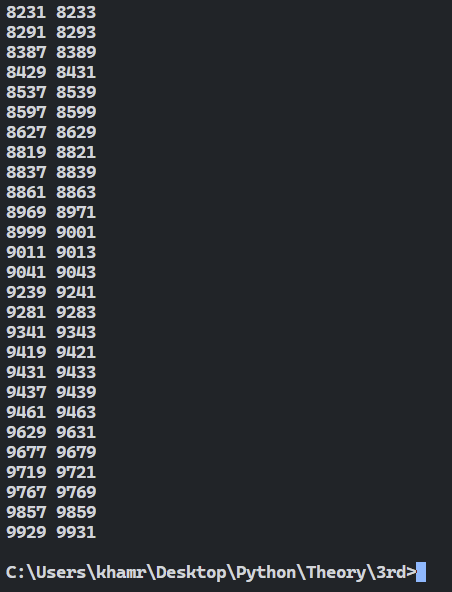
            break

    if is\_prime and is\_twin\_prime:

        print(num, twin\_num)

**Output:**

****

****

**10. To find out the list of months with a Friday the 13th for any given year.**

**Source code:**

**Output:**

**11.** **To draw a Pascal triangle based on given number of rows Hints for drawing the Pascal’s Triangle The first 6 rows of Pascal’s Triangle are shown in Figure-1: Let us learn how to Build the Pascal’s Triangle manually. • At the top center of a piece of paper let us write the number “1.” • On the next row we write two 1’s so that they form a triangle. • On each of the subsequent rows we start and end with 1’s and compute each interior term by adding the two numbers above it.**

****

**Source code:**

n = int(input("Enter the number of rows: "))

triangle = [[1]]

for i in range(1, n):

    row = [1]

    for j in range(1, i):

        row.append(triangle[i - 1][j - 1] + triangle[i - 1][j])

    row.append(1)

    triangle.append(row)

for i in range(n):

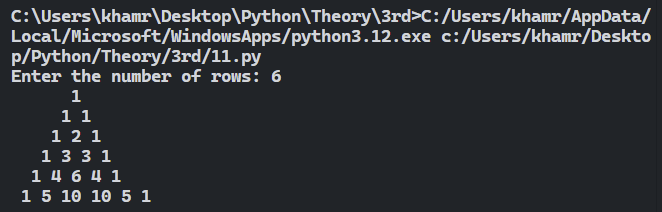
    print(' ' \* (n - i - 1), end=' ')

    for j in range(i + 1):

        print(triangle[i][j], end=' ')

    print()

**Output:**

****

**12.** **A program is required which accepts two numbers then prints out a random number somewhere between those two numbers without using the built in random function.**

**Source code:**

**Output:**

**13. A program is required which will accept a date in either long hand or short hand (e.g. either 19/7/2019 or 19th July, 2019 ) with a view to determine if the given date is valid. e.g. 14/13/2019 is clearly not valid.**

**Source code:**

date\_input = input("Enter the date (e.g., 19/7/2019 or 19th July, 2019): ")

long\_hand\_suffixes = ["st", "nd", "rd", "th"]

long\_hand\_months = ["january", "february", "march", "april", "may", "june",

                    "july", "august", "september", "october", "november", "december"]

for suffix in long\_hand\_suffixes:

    date\_input = date\_input.replace(suffix, "")

date\_input = date\_input.replace(",", "")

parsed\_date = None

parts = date\_input.split()

if len(parts) == 3:

    day = int(parts[0])

    month\_str = parts[1].lower()

    year = int(parts[2])

    if month\_str in long\_hand\_months:

        month = long\_hand\_months.index(month\_str) + 1

        parsed\_date = (day, month, year)

else:

    parts = date\_input.split('/')

    if len(parts) == 3:

        day = int(parts[0])

        month = int(parts[1])

        year = int(parts[2])

        parsed\_date = (day, month, year)

if parsed\_date:

    day, month, year = parsed\_date

    valid\_date = True

    if month < 1 or month > 12:

        valid\_date = False

    if day < 1:

        valid\_date = False

    if month in [1, 3, 5, 7, 8, 10, 12] and day > 31:

        valid\_date = False

    if month in [4, 6, 9, 11] and day > 30:

        valid\_date = False

    if month == 2:

        if (year % 4 == 0 and (year % 100 != 0 or year % 400 == 0)) and day > 29:

            valid\_date = False

        if (year % 4 != 0 or (year % 100 == 0 and year % 400 != 0)) and day > 28:

            valid\_date = False

    if valid\_date:

        print("The date is valid.")

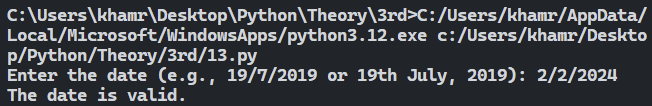
    else:

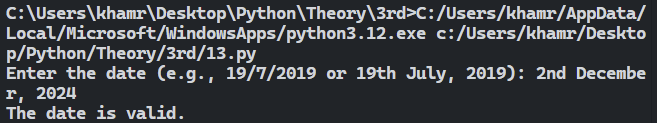
        print("The date is not valid.")

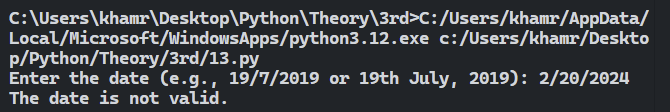
else:

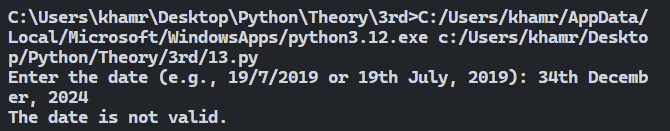
    print("The date format is not recognized.")

**Output:**

****

****

****

****

**14 a) A Harshad number is a number that is divisible by the sum of its own digits. For example, 1729 is a Harshad number because 1 + 7 + 2 + 9 = 19 and 1729 = 1991. Harshad numbers are also known as Niven numbers.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

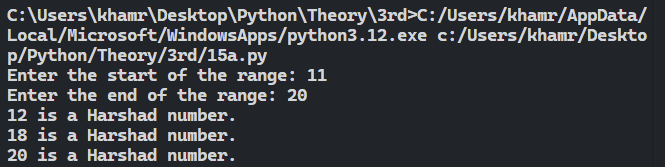
for num in range(start, end + 1):

sum\_of\_digits = sum(int(digit) for digit in str(num))

if num % sum\_of\_digits == 0:

print(f"{num} is a Harshad number.")

**Output:**

****

**14 b) An Abundant number is a number that is smaller than the sum of its aliquot parts (Proper divisors). For example, twelve is the smallest abundant number, because, the sum of its aliquot parts is 1+2+3+4+6=16—-followed by 18, 20, 24 and 30.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

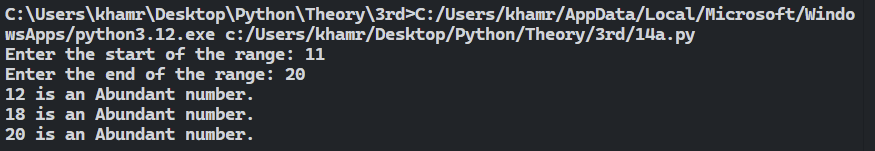
divisors = [i for i in range(1, num) if num % i == 0]

sum\_of\_divisors = sum(divisors)

if sum\_of\_divisors > num:

print(f"{num} is an Abundant number.")

**Output:**

****

**14 c) A weird number is an abundant number that is not semi perfect; in other words, n is weird if the sum of its divisors is greater than n, but n is not equal to the sum of any subset of its divisors. The first few weird numbers are 70, 836, 4030, 5830, and 7192.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

    divisors = [i for i in range(1, num) if num % i == 0]

    sum\_of\_divisors = sum(divisors)

    if sum\_of\_divisors > num:

        found = False

        for i in range(1, 1 << len(divisors)):  # Check all subsets of divisors

            subset\_sum = 0

            for j in range(len(divisors)):

                if i & (1 << j):

                    subset\_sum += divisors[j]

            if subset\_sum == num:

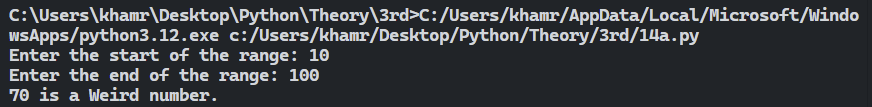
                found = True

                break

        if not found:

            print(f"{num} is a Weird number.")

**Output:**

****

**14 d) Amicable numbers are pairs of numbers, also known as friendly numbers, each of whose aliquot parts add to give the other number. (An aliquot part is any divisor that doesn’t include the number itself).**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

    divisors = [i for i in range(1, num) if num % i == 0]

    sum\_of\_divisors = sum(divisors)

    for other\_num in range(num + 1, end + 1):  # Check numbers greater than current num

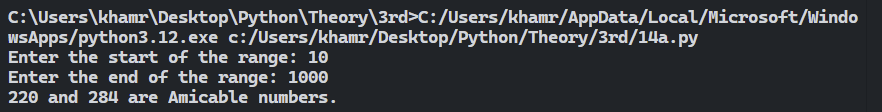
        divisors\_of\_other = [i for i in range(1, other\_num) if other\_num % i == 0]

        sum\_of\_other\_divisors = sum(divisors\_of\_other)

        if sum\_of\_divisors == other\_num and sum\_of\_other\_divisors == num:

            print(f"{num} and {other\_num} are Amicable numbers.")

**Output:**

****

**14 e) An Automorphic number, also known as an automorph, is a number n whose square ends in n. For instance 5 is automorphic, because 5 2 = 25, which ends in 5. A number n is called trimorphic if n 3 ends in n. For example 493 , = 117649, is trimorphic. Not all trimorphic numbers are automorphic. A number n is called tri-automorphic if 3n 2 ends in n; for example 667 is tri-automorphic because 3 × 6672 , = 1334667, ends in 667.**

**Source code:**

start = int(input("Start range: "))

end = int(input("End range: "))

for n in range(start, end + 1):

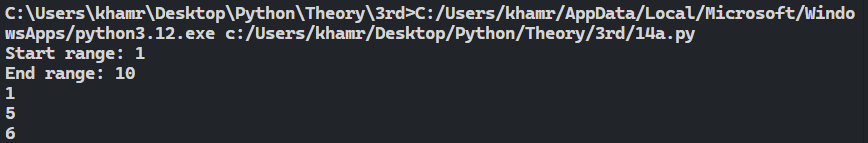
square = n \* n

length = len(str(n))

if str(square)[-length:] == str(n):

print(n)

**Output:**

****

**14 f) A Fermat number is a number defined by the formula Fn = 2 2 n + 1 numbers, F0 = 3, F1 = 5, F2 = 17, F3 = 257, and F4 = 65,537. It is required to test whether all the Fermat numbers are prime or not.**

**Source code:**

start = int(input("Enter start term: "))

end = int(input("Enter end term: "))

for n in range(start, end + 1):

F\_n = 2 \*\* (2 \*\* n) + 1

is\_prime = True

for i in range(2, F\_n):

if F\_n % i == 0:

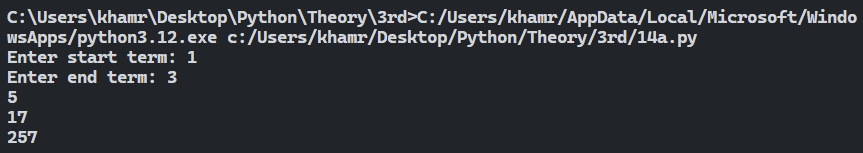
is\_prime = False

break

if is\_prime:

print(F\_n)

**Output:**

****

**14 g) A Happy number is one which ultimately generates 1 through the iterative process of summing of the squares of the digits.For example,7 → (72) → 49 → (42 +92) → 97 → (92 + 72) → 130 → (12 + 32) → 101. So, 7 is a Happy number. But, 20 is not a happy number, since,**

**22+02=4**

**42 = 16**

**12+62=37**

**32 +72-58**

**52 +82-89**

**82 +92 =145**

**12 +42+52=40**

**42+02=4(Reappears a number obtained earlier)**

**Source code:**

start = eval(input("Enter start range:"))

end = eval(input("Enter end range:"))

for n in range(start, end + 1):

    current = n

    seen\_numbers = []

    while current != 1 and current not in seen\_numbers:

        seen\_numbers.append(current)

        sum\_of\_squares = 0

        while current > 0:

            digit = current % 10

            sum\_of\_squares += digit \* digit

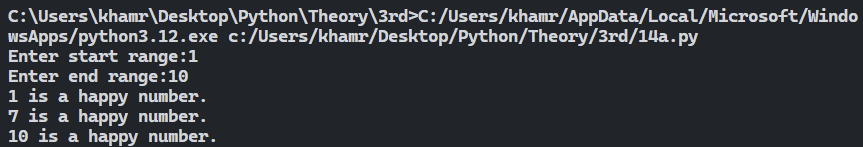
            current //= 10

        current = sum\_of\_squares

    if current == 1:

        print(n,"is a happy number.")

**Output:**

****

**14 h) A special two-digit number is such that when the sum of its digits is added to the product of its digits, the result equals the original two-digit number. For example, for the number 59,Sum of the digits=5+9=14 and product of its digits=5\*9=45 and hence total=14+45=59.**

**Source code:**

start = eval(input("Starting range(10-99):"))

end = eval(input("Ending range(10-99):"))

print("The special two digits num in the range:")

for num in range(start, end + 1):

tens = num // 10

units = num % 10

sum\_digits = tens + units

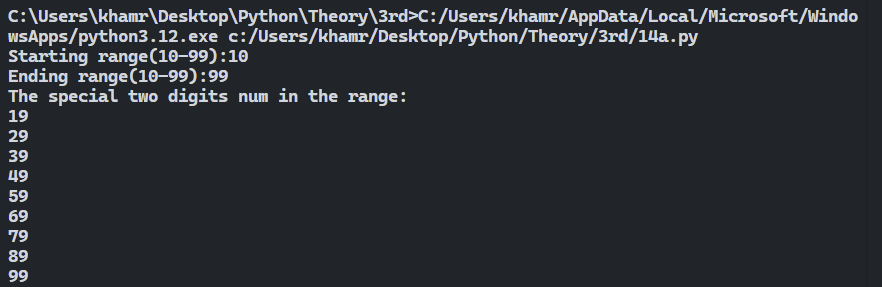
product\_digits = tens \* units

result = sum\_digits + product\_digits

if result == num:

print(num)

**Ouput:**

****

**14 i) WAP to accept a number and then to check whether it is a perfect square or not. If it is a perfect square then show a message for that; if not, then find out the least number to be added to the number to make the given number a perfect square. For example, if the given number is 1950, then the least number to be added to the number to make it a perfect square is determined as follows: 452 - 1950=75.**

**Source code:**

num = eval(input("Enter a number: "))

sqrt\_num = int(num \*\* 0.5)

if sqrt\_num \* sqrt\_num == num:

    print("The number is a perfect square.")

else:

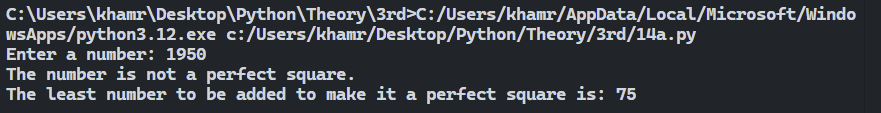
    print("The number is not a perfect square.")

    next\_perfect\_square = (sqrt\_num + 1) \* (sqrt\_num + 1)

    diff = next\_perfect\_square - num

    print(f"The least number to be added to make it a perfect square is: {diff}")

**Output:**

****

**14 j) A number is said to be a Niven number if it is divisible by the sum of its digits. For example,126 is a Niven number.**

**Source code:**

start\_range = int(input("Enter the starting range: "))

end\_range = int(input("Enter the ending range: "))

for num in range(start\_range, end\_range + 1):

    sum\_digits = 0

    temp = num

    while temp > 0:

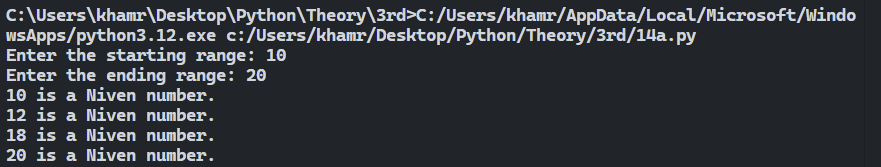
        sum\_digits += temp % 10

        temp //= 10

    if sum\_digits != 0 and num % sum\_digits == 0:

        print(num, "is a Niven number.")

**Output:**

****

**14 k) A number is said to be a Neon number if the sum of the digits of the square of the number equals the number. For example, 9 is a Neon number.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

    square = num \* num

    sum\_digits = 0

    while square > 0:

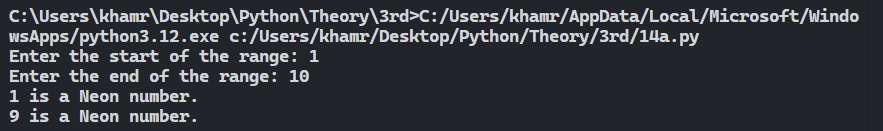
        sum\_digits += square % 10

        square //= 10

    if sum\_digits == num:

        print(num, "is a Neon number.")

**Output:**

****

**14 l) A number is said to be an Automorphic number if the number is contained in the ending digits of its square. For example, 25 is such a number because 252 =625 where 25 the original number is contained.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

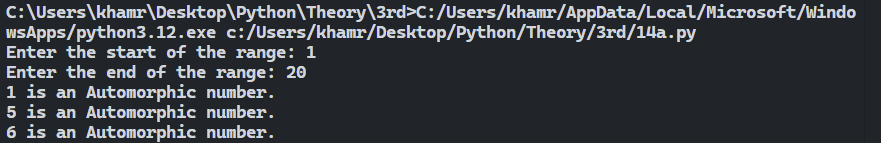
for num in range(start, end + 1):

square = num \* num

if str(square).endswith(str(num)):

print(num, "is an Automorphic number.")

**Output:**

****

**14 m) Two numbers are said to be co-prime if their HCF is one.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

print("Co-prime numbers in range are:")

for num1 in range(start, end + 1):

for num2 in range(num1 + 1, end + 1):

min\_num = min(num1, num2)

hcf = 1

for i in range(2, min\_num + 1):

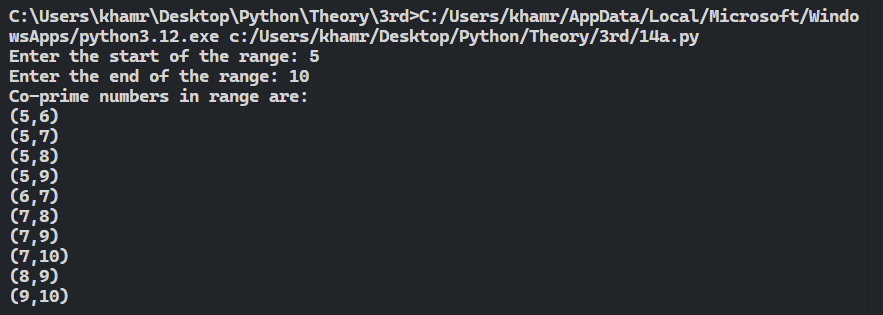
if num1 % i == 0 and num2 % i == 0:

hcf = i

if hcf == 1:

print(f"({num1},{num2})")

**Output:**

****

**14 n) WAP to display the product of the successors of even digits of a given number.**

**Source code:**

num = eval(input("Enter a number: "))

product = 1

temp = num

while temp > 0:

    digit = temp % 10

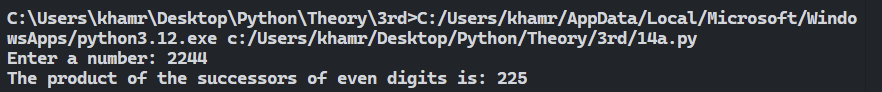
    if digit % 2 == 0:

        product \*= (digit + 1)

    temp //= 10

print("The product of the successors of even digits is:", product)

**Output:**

****

**14 O) A number is said to be a Pronic number if it a product of two consecutive integers. Example: 12, 20, 42 etc. such numbers are also known as Oblong numbers.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

is\_pronic = False

for i in range(1, num + 1):

if i \* (i + 1) == num:

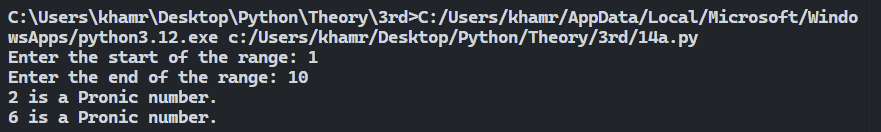
is\_pronic = True

break

if is\_pronic:

print(num, "is a Pronic number.")

**Output:**

****

**14 p) A prime number is said to a Twisted Prime, if the new number formed by reversing its digits is also a prime number. For example, 167 is a twisted prime because 761 is also prime.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

is\_prime = True

for i in range(2, num):

if num % i == 0:

is\_prime = False

break

if is\_prime:

reversed\_num = int(str(num)[::-1])

is\_twisted\_prime = True

for i in range(2, reversed\_num):

if reversed\_num % i == 0:

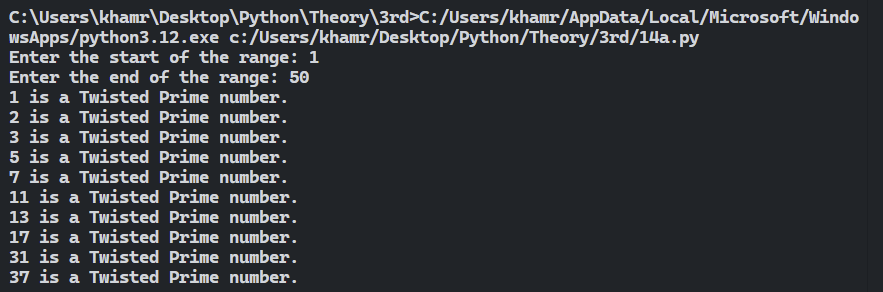
is\_twisted\_prime = False

break

if is\_twisted\_prime:

print(num, "is a Twisted Prime number.")

**Output:**

****

**14 q)** **A number is said to a Duck number if it contains a zero.**

**Source code:**

start = eval(input("Enter the start of the range: "))

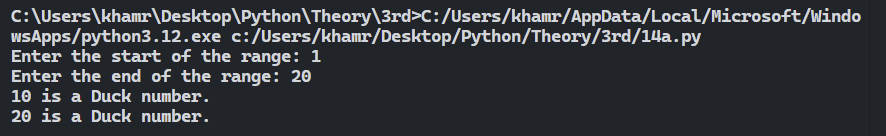
end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

if '0' in str(num)[1:]:

print(num, "is a Duck number.")

**Output:**

****

**14 r) A number is said to be an Abundant number if the sum of its proper divisors is greater than the number itself. For example, 12 is an abundant number.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

sum\_divisors = 0

for i in range(1, num):

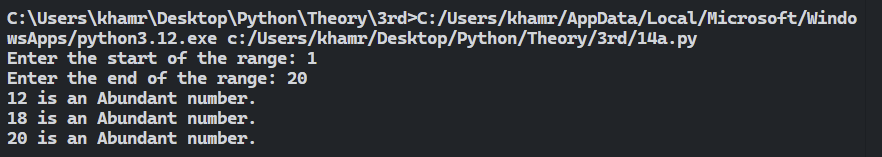
if num % i == 0:

sum\_divisors += i

if sum\_divisors > num:

print(num, "is an Abundant number.")

**Output:**

****

**14 s) A number is said to a Spy number if the sum of its digits equals the product of its digits. For example, 1124 is a Spy number.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

sum\_digits = 0

product\_digits = 1

temp = num

while temp > 0:

digit = temp % 10

sum\_digits += digit

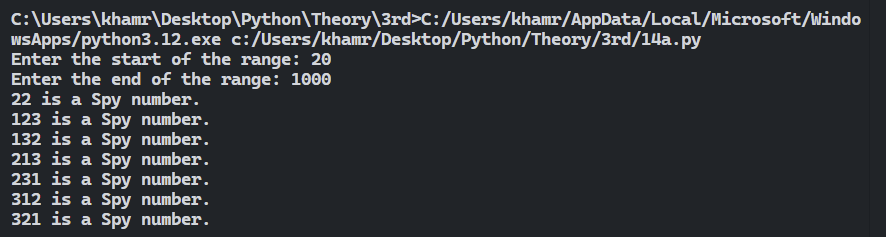
product\_digits \*= digit

temp //= 10

if sum\_digits == product\_digits:

print(num, "is a Spy number.")

**Output:**

****

**14 t) A number is said to be a Harshad number, if it is divisible by the sum of its digits. For example, 132 is a Harshad number.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

for num in range(start, end + 1):

sum\_digits = 0

temp = num

while temp > 0:

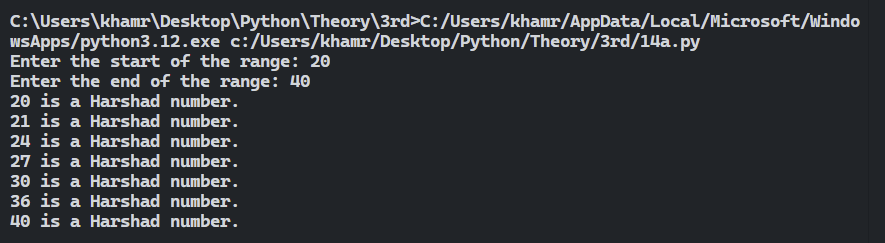
sum\_digits += temp % 10

temp //= 10

if num % sum\_digits == 0:

print(num, "is a Harshad number.")

**Output:**

****

**14 u) A number is said to be a Sunny number if the square root of its successor is a whole number.**

**Source code:**

start = eval(input("Enter the start of the range: "))

end = eval(input("Enter the end of the range: "))

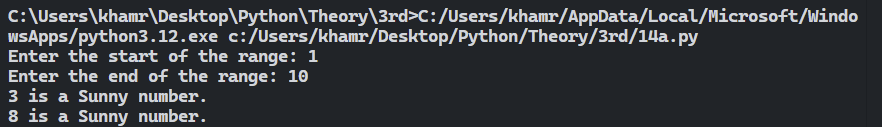
for num in range(start, end + 1):

next\_num = num + 1

if (next\_num \*\* 0.5).is\_integer():

print(num, "is a Sunny number.")

**Output:**

****

**14 v) Tribonacci numbers are a sequence of numbers similar to Fibonacci numbers except that such a number is formed by adding the three preceding numbers. For example, 1, 1, 2, 4, 7,13, . . . . . . . . . ..are Tribonacci numbers.**

**Source code:**

srand = eval(input("Enter the start of the range: "))

erand = eval(input("Enter the end of the range: "))

print(f"The tribonacci numbers between {srand} and {erand} are:")

a = 1

b = 1

c = 2

if srand <= a <= erand:

print(a)

if srand <= b <= erand:

print(b)

if srand <= c <= erand:

print(c)

while True:

d = a + b + c

if d > erand:

break

if d >= srand:

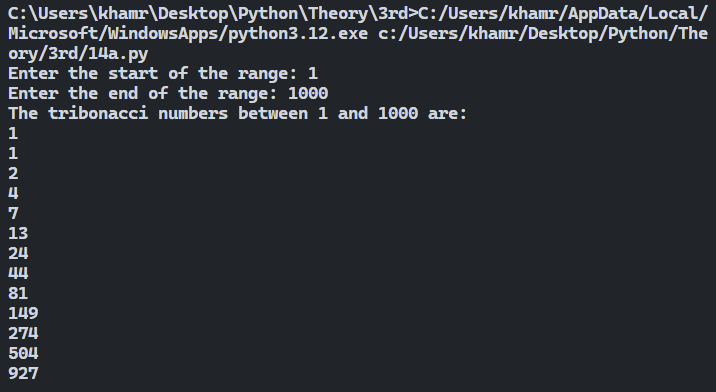
print(d)

a = b

b = c

c = d

**Output:**

****

**14 w) i) The sum of given n terms for some given value of x(where there is x) is required for the following series: i. 1+ (1+2) +(1+2+3) + · · · +(1+2+3+· · ·n).**

**Source code:**

n = eval(input("Enter the value of n: "))

total\_sum = 0

for i in range(1, n+1):

inner\_sum = 0

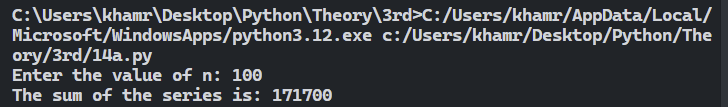
for j in range(1, i+1):

inner\_sum += j

total\_sum += inner\_sum

print("The sum of the series is:", total\_sum)

**Output:**

****

**14 w) ii) 1+(1\*2) +(1\*2\*3)+· · ·+(1\*2\*3\*· · · \*n).**

**Source code:**

n = eval(input("Enter the value of n: "))

total\_sum = 0

for i in range(1, n + 1):

product = 1

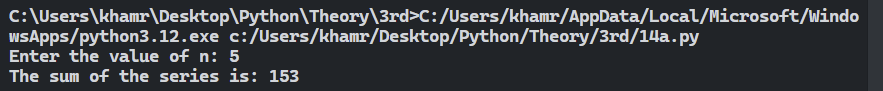
for j in range(1, i + 1):

product \*= j

total\_sum += product

print("The sum of the series is:", total\_sum)

**Output:**

****

**14 w) iii) 1+1+2/1\*2+1+2+3/1\*2\*3+…….+1+2+3+…+n/1\*2\*3\*…..\*n up to n terms.**

**Source code:**

n = eval(input("Enter n: "))

series\_sum = 0

for k in range(1, n + 1):

fact\_k = 1

for j in range(1, k + 1):

fact\_k \*= j

numerator = 0

for m in range(1, k + 1):

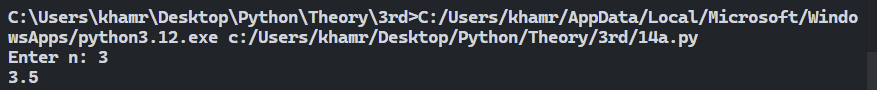
numerator += m

term = numerator / fact\_k

series\_sum += term

print(series\_sum)

**Output:**

****

**14 x) A number is said to be a Multiple Harshed Number if a Harshed Number, when divided by the sum of its digits, produces another Harshed Number. For example, 6804 is a Multiple Harshed Number.**

**Source code:**

srand = eval(input("Enter the start of the range: "))

erand = eval(input("Enter the end of the range: "))

for num in range(srand, erand + 1):

digit\_sum = 0

temp = num

while temp > 0:

digit\_sum += temp % 10

temp //= 10

if num % digit\_sum == 0:

quotient = num // digit\_sum

quotient\_digit\_sum = 0

temp = quotient

while temp > 0:

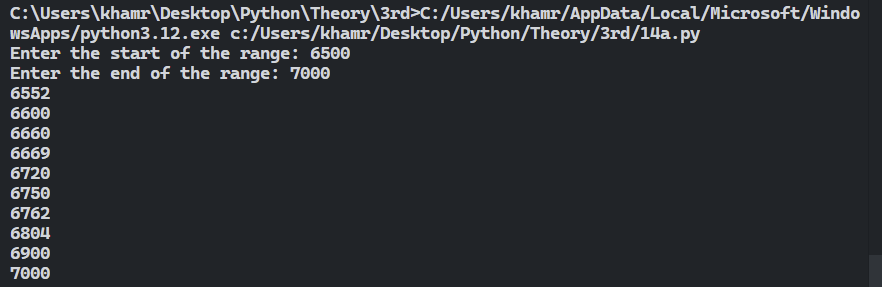
quotient\_digit\_sum += temp % 10

temp //= 10

if quotient % quotient\_digit\_sum == 0:

print(num)

**Output:**

****

**14 y) A number is said to be a Magic Number if the eventual sum of the digits of the number becomes 1. For example, 55 and 289 are magic numbers because, for 55,5+5=10, 1+0=1, similarly for 289,2+8+9=19,1+9=10,1+0=1.**

**Source code:**

srand = eval(input("Enter the start of the range: "))

erand = eval(input("Enter the end of the range: "))

for num in range(srand, erand + 1):

temp = num

while temp > 9:

digit\_sum = 0

while temp > 0:

digit\_sum += temp % 10

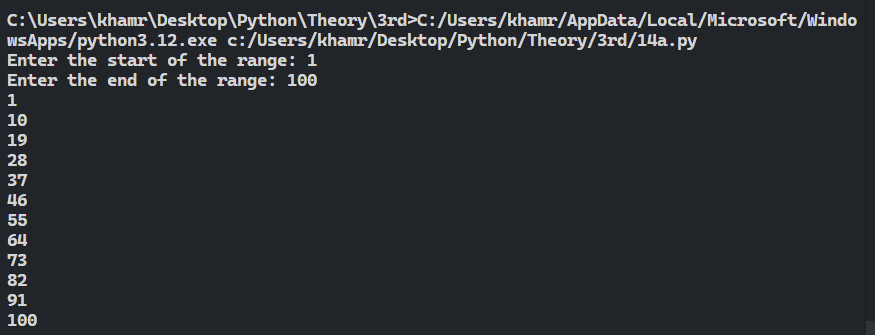
temp //= 10

temp = digit\_sum

if temp == 1:

print(num)

**Output:**

****

**15 a) A right-angled triangle of consecutive natural numbers from 1 to n is called a Floyd’s triangle.**

**Source code:**

**Output:**

**15 b) A hyperfactorial number is a number such as 108, which is equal to 3 3 ×2 2 ×1 1 . In general, the n th hyperfactorial H(n) is given by: H(n) = n n (n − 1)(n−1) · · · 3 32 211.**

**Source code:**

n = int(input("Enter a number: "))

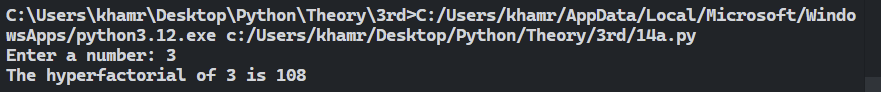
result = 1

for i in range(1, n + 1):

result \*= i \*\* i

print("The hyperfactorial of", n, "is", result)

**Output:**

****

**15 c) A Kaprekar number of d digits is defined as a number the square of which if separated into two parts of d and d / (d1) digits depending on the availability of digits, can regenerate the number by the sum of the two parts. For example: 9 is a Kaprekar number; because, 9 2=81 and as 9 is a one-digit number, we separate the digits in the square in two parts each containing one digit i.e. 8 and 1. Again, 8+1=9, regenerates the number.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

square = str(num \*\* 2)

length = len(square)

for i in range(1, length):

left\_part = int(square[:i])

right\_part = int(square[i:])

if right\_part == 0:

continue

if left\_part + right\_part == num:

print(num, "is a Kaprekar number")

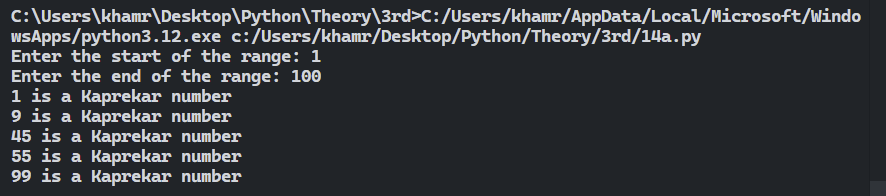
break

else:

if int(square) == num:

print(num, "is a Kaprekar number")

**Output:**

****

**15 d) A vampire number is a natural number x that can be factorized as y × z in such a way that the number of occurrences of a particular digit in the representation of x in a given base (say 10) appears the same number of times in the representations in that same base of y and z together. For example, 2187 is a vampire number since 2187 = 21 × 87 ; similarly 136948 is a vampire because 136948 = 146 × 938. Vampire numbers are a whimsical idea that was introduced by Clifford Pickover in 1995**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

    num\_str = str(num)

    num\_len = len(num\_str)

    if num\_len % 2 != 0:

        continue

    half\_len = num\_len // 2

    found\_vampire = False

    for i in range(10 \*\* (half\_len - 1), 10 \*\* half\_len):

        for j in range(10 \*\* (half\_len - 1), 10 \*\* half\_len):

            if i \* j == num:

                fangs = str(i) + str(j)

                fangs\_sorted = sorted(fangs)

                num\_sorted = sorted(num\_str)

                if fangs\_sorted == num\_sorted:

                    print(num, "is a vampire number with fangs", i, "and", j)

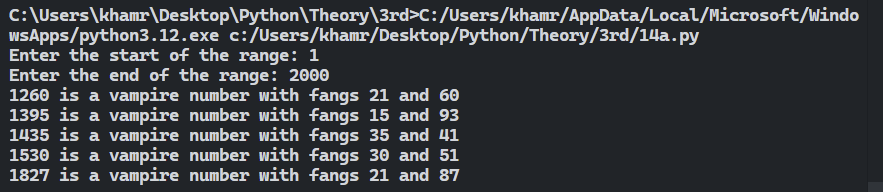
                    found\_vampire = True

                    break

        if found\_vampire:

            break

**Output:**

****

**15 e) A sublime number is a number such that both the sum of its divisors and the number of its divisors are perfect numbers. The smallest sublime number is 12. There are 6 divisors of 12 – 1, 2, 3, 4, 6, and 12 – the sum of which is 28. Both 6 and 28 are perfect. The second sublime number begins 60865..., ends ...91264, and has a total of 76 digits! It is not known if there are larger even sublime numbers, nor if there are any odd sublime numbers.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

for num in range(start, end + 1):

s = 0

f = 0

s1 = 0

s2 = 0

for i in range(1, num + 1):

if num % i == 0:

s += i

f += 1

for j in range(1, s):

if s % j == 0:

s1 += j

for j in range(1, f):

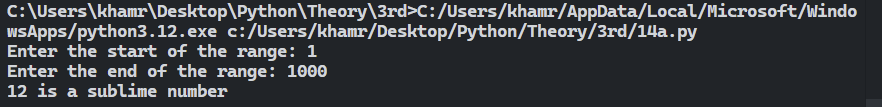
if f % j == 0:

s2 += j

if s1 == s and s2 == f:

print(num, "is a sublime number")

**Output:**

****

**15 f) A Smith number is a composite number, the sum of whose digits equals the sum of the digits of its prime factors. The name stems from a phone call in 1984 by the mathematician Albert Wilansky to his brother-in-law, called Smith, during which Wilansky noticed that the phone number, 4937775, obeyed the condition just mentioned. Specifically: 4937775 = 3 × 5 × 5 × 65837.**

**Source code:**

start = int(input("Enter the start of the range: "))

end = int(input("Enter the end of the range: "))

def is\_prime(n):

if n <= 1:

return False

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

if n % i == 0:

return False

return True

for num in range(start, end + 1):

if is\_prime(num):

continue

sum\_of\_digits = sum(int(digit) for digit in str(num))

temp\_num = num

prime\_factors\_digits\_sum = 0

for i in range(2, temp\_num + 1):

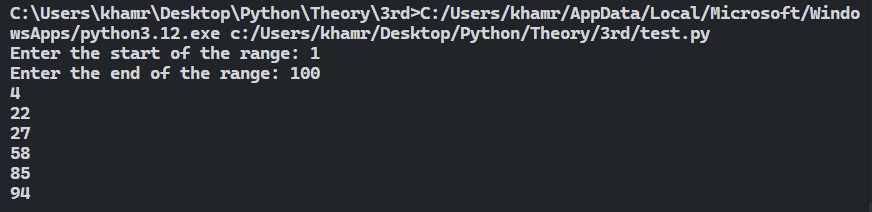
while temp\_num % i == 0:

prime\_factors\_digits\_sum += sum(int(digit) for digit in str(i))

temp\_num //= i

if sum\_of\_digits == prime\_factors\_digits\_sum:

print(num)

**Output:**

**15 g) A lucky number is a number in a sequence, first identified and named around 1955 by Stanislaw Ulam, that evades a particular type of number "sieve" (similar to the famous Sieve of Eratosthenes), which works as follows. Start with a list of integers, including 1, and cross out every second number: 2, 4, 6, 8, ... The second surviving integer is 3. Cross out every third number not yet eliminated. This removes 5, 11, 17, 23, ... The third surviving number from the left is 7; cross out every seventh integer not yet eliminated: 19, 39, ... Repeat this process indefinitely and the numbers that survive are the "lucky" ones:**

**Source code:**

start\_range = eval(input("Enter start range:"))

end\_range = eval(input("Enter end range:"))

print("Lucky number in range:")

for x in range(start\_range, end\_range + 1):

    n = x

    counter = 2

    is\_lucky = True

    while counter <= n:

        if n % counter == 0:

            is\_lucky = False

            break

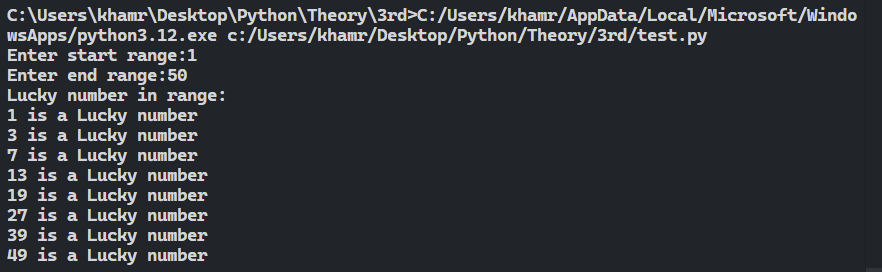
        n -= n // counter

        counter += 1

    if is\_lucky:

        print(x, "is a Lucky number")

**Output:**

****

**15 h) A Pandigital number is an integer that contains each of the digits from 0 to 9 and whose leading digit is nonzero. The first few pandigital numbers are 1023456789, 1023456798, 1023456879, 1023456897, and 1023456978. A ten-digit pandigital number is always divisible by 9. If zeros are excluded, the first few "zeroless" pandigital numbers are 123456789, 123456798, 123456879, 123456897, 123456978, and 123456987, and the first few zeroless pandigital primes are 1123465789, 1123465879, 1123468597, 1123469587, and 1123478659. The sum of the first 32423 (a palindromic number) consecutive primes is 5897230146, which is pandigital. No other palindromic number shares this property. Examples of palindromic numbers that are the product of pandigital numbers are 2 970 408 257 528 040 792 (= 1 023 687 954 × 2 901 673 548) and 5 550 518 471 748 150 555 (= 1 023 746 895 × 5 421 768 309), both found in 2001. A pandigital product is a product in which the digits of the multiplicand, multiplier, and product, taken together, form a pandigital number.**

**Source code:**

start\_range = eval(input("Enter start range:"))

end\_range = eval(input("Enter end range:"))

print("Pandigital numbers in given range:")

for num in range(start\_range, end\_range + 1):

    num\_str = str(num)

    digits = "0123456789"

    is\_pandigital = True

    for digit in digits:

        if digit not in num\_str:

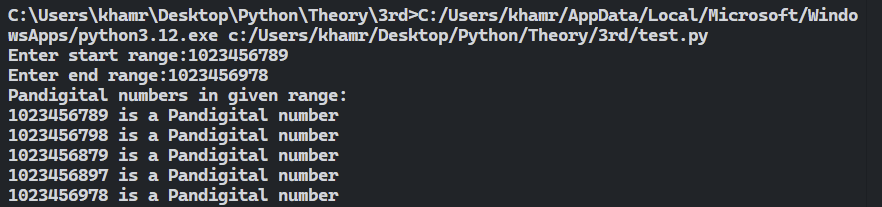
            is\_pandigital = False

            break

    if is\_pandigital:

        print(num, "is a Pandigital number")

**Output:**

****