

## Part2AssignmentAnswers

January 28, 2026

```
[ ]: # given a number, program to find sum of largest prime factors of each 9 consecutive numbers starting from that number

def find_factors(num):
    factors=[]
    for i in range(2,num+1):
        if num%i==0:
            factors.append(i)
    return factors

def is_prime(num,i):
    if(i==1):
        return True
    elif (num%i==0):
        return False
    else:
        return is_prime(num,i-1)

def find_largest_prime_factor(list_of_factors):
    largest_prime_factor=0
    for factor in list_of_factors:
        if is_prime(factor,factor-1):
            if factor>largest_prime_factor:
                largest_prime_factor=factor
    return largest_prime_factor

def find_f(num):
    # return largest prime factor of num
    factors=find_factors(num)
    largest_prime_factor=find_largest_prime_factor(factors)
    return largest_prime_factor

def find_g(num):
    # return sum of largest prime factors of each 9 consecutive numbers starting from num
    total_sum=0
    for i in range(num,num+9):
```

```

        total_sum+=find_f(i)
    return total_sum

print(find_g(10))  # Example usage

```

66

```
[ ]: # find duplicates and print (order should be same as input list)
def find_duplicates(list_of_numbers):
    duplicates=[]
    seen=set()
    for number in list_of_numbers:
        if number in seen and number not in duplicates:
            duplicates.append(number)
        else:
            seen.add(number)
    return duplicates
```

[3]: # check anagram

```

def are_anagrams(str1, str2):
    s1=str1.lower()
    s2=str2.lower()

    if len(s1)!=len(s2):
        return False
    if sorted(s1)!=sorted(s2):
        return False

    # no same character at same position
    for i in range(len(s1)):
        if s1[i]==s2[i]:
            return False
    return True

print(are_anagrams("Listen","Silent")) # False, i is at same position in both
                                         ↴strings

```

False

```
[2]: # remove duplicate characters from a string
def remove_duplicates(s):
    result=""
    for char in s:
        if char not in result:
            result+=char
    return result
```

```
print(remove_duplicates("1122334455ababzzz@@@123##**"))
```

```
12345abz@#*
```

```
[4]: # check whether a given number is perfect number or not
# A perfect number is a positive integer that is equal to the sum of its proper
# positive divisors, excluding the number itself.

def check_perfect_number(n):
    if n<1:
        return False
    sum_divisors=0
    for i in range(1,n):
        if n%i==0:
            sum_divisors+=i
    return sum_divisors==n

def check_perfectno_from_list(no_list):
    perfect_numbers=[]
    for num in no_list:
        if check_perfect_number(num):
            perfect_numbers.append(num)
    return perfect_numbers

print(check_perfectno_from_list([6,28,12,15,496,8128,10])) # [6, 28, 496, 8128]
```

```
[6, 28, 496, 8128]
```

```
[ ]: # Write a python program to help an airport manager to generate few statistics
# based on the ticket details available for a day.

# lex_auth_0127382193364008961449

# Sample ticket list - ticket format: "flight_no:source:destination:ticket_no"
# Note: flight_no has the following format - "airline_name followed by three
# digit number

# Global variable
ticket_list = [
    "AI567:MUM:LON:014",
    "AI077:MUM:LON:056",
    "BA896:MUM:LON:067",
    "SI267:MUM:SIN:145",
    "AI077:MUM:CAN:060",
    "SI267:BLR:MUM:148",
    "AI567:CHE:SIN:015",
    "AI077:MUM:SIN:050",
    "AI077:MUM:LON:051",
```

```

"SI267:MUM:SIN:146",
]

def find_passengers_flight(airline_name="AI"):
    # This function finds and returns the number of passengers travelling in
    ↪the specified airline.
    count = 0
    for i in ticket_list:
        string_list = i.split(":")
        if string_list[0].startswith(airline_name):
            count += 1
    return count

def find_passengers_destination(destination):
    # Write the logic to find and return the number of passengers traveling to
    ↪the specified destination
    count = 0
    for i in ticket_list:
        string_list = i.split(":")
        if string_list[2] == destination:
            count += 1
    return count

def find_passengers_per_flight():
    """Write the logic to find and return a list having number of passengers
    ↪traveling per flight based on the details in the ticket_list
    In the list, details should be provided in the format:
    [flight_no:no_of_passenger, flight_no:no_of_passenger, etc.]."""
    passenger_count = {}
    for ticket in ticket_list:
        flight_no = ticket.split(":")[0]
        if flight_no in passenger_count:
            passenger_count[flight_no] += 1
        else:
            passenger_count[flight_no] = 1
    result = [f"{flight}:{count}" for flight, count in passenger_count.items()]
    return result

def sort_passenger_list():
    # Write the logic to sort the list returned from
    ↪find_passengers_per_flight() function in the descending order of number of
    ↪passengers
    passenger_list = find_passengers_per_flight()

```

```

passenger_list.sort(key=lambda x: int(x.split(":")[1]), reverse=True)
return passenger_list

# Provide different values for airline_name and destination and test your ↴
↪program.
print(find_passengers_flight("AI"))
print(find_passengers_destination("LON"))
print(sort_passenger_list())

```

```
[ ]: # nearest palindrome
def nearest_palindrome(number):
    num=number+1
    while True:
        if str(num)==str(num)[::-1]:
            return num
        num+=1
print(nearest_palindrome(123)) #121
```

121

```
[ ]: # circular prime(ex: 197--> 197, 971, 719 all are prime)
def check_prime(number):
    if number<2:
        return False
    for i in range(2,int(number**0.5)+1):
        if number%i==0:
            return False
    return True

def rotations(num):
    s=str(num)
    result=[]
    for i in range(len(s)):
        rotated=s[i:]+s[:i]
        result.append(int(rotated))
    return result

def get_circular_prime_count(limit):
    count=0
    for num in range(2,limit):
        is_circular_prime=True
        for rot in rotations(num):
            if not check_prime(rot):
                is_circular_prime=False
                break
        if is_circular_prime:
```

```

        count+=1
    return count
print(get_circular_prime_count(100)) # 13

```

```

[6]: # Validate details provided by a user as a part of registering to a web application

# Write a function validate_name(name) to validate the user name
#     Name should not be empty, name should not exceed 15 characters
#     Name should contain only alphabets

# Write a function validate_phone_no(phono) to validate the phone number
#     Phone number should have 10 digits
#     Phone number should not have any characters or special characters
#     All the digits of the phone number should not be same.
#     Example: 9999999999 is not a valid phone number

# Write a function validate_email_id(email_id) to validate email Id
#     It should contain one '@' character and '.com'
#     '.com' should be present at the end of the email id.
#     Domain name should be either 'gmail', 'yahoo' or 'hotmail'
# Note: Consider the format of email id to be username@domain_name.com

import re
def validate_name(name):
    pattern="^ [A-Za-z] {1,15} $"
    if re.match(pattern,name):
        return True
    return False

def validate_phone_no(phno):
    pattern="^ [0-9] {10} $"
    if re.match(pattern,phno):
        if len(set(phno))==1:
            return False
        return True
    return False

def validate_email_id(email_id):
    pattern="^ [A-Za-z0-9. _%+-]+ @ (gmail| yahoo | hotmail) \. com $"
    if re.match(pattern,email_id):
        return True
    return False

def validate_all(name,phone_no,email_id):
    if validate_name(name) and validate_phone_no(phone_no) and validate_email_id(email_id):

```

```
        return True
    return False

print(validate_all("JohnDoe", "9876543210", "john.doe@gmail.com"))
```

True

```
[7]: # Luhn algorithm to validate credit card numbers
def validate_credit_card_number(card_number):
    card=str(card_number)
    if not card.isdigit() or len(card)<13 or len(card)>19:
        return False
    total=0
    reverse_digits=card[::-1]
    for i,digit in enumerate(reverse_digits):
        n=int(digit)
        if i%2==1:
            n=n*2
            if n>9:
                n=n-9
        total+=n
    return total%10==0
print(validate_credit_card_number(4532015112830366)) # True
```

True