**DAY 6 -**PROGRAMMING FUNDAMENTALS USING PYTHON

Assignment 40:

Write a recursive function, **is\_palindrome()** to find out whether a string is a palindrome or not. The function should return true, if it is a palindrome. Else it should return false.  
  
**Note-** Perform case insensitive operations wherever necessary.

#PF-Assgn-40

def is\_palindrome(word):

#Remove pass and write your logic here

return word==word[::-1]

#Provide different values for word and test your program

result=is\_palindrome("MadAMa")

if(result):

print("The given word is a Palindrome")

else:

print("The given word is not a Palindrome")

Assignment 41:

A 10-substring of a number is a substring of its digits that sum up to 10.  
  
For example, the 10-substrings of the number 3523014 are:  
**352**3014, 3**523**014, 3**5230**14, 35**23014**  
  
Write a python function, **find\_ten\_substring(num\_str)** which accepts a string and returns the list of 10-substrings of that string.  
  
Handle the possible errors in the code written inside the function.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| '3523014' | ['5230', '23014', '523', '352'] |

#PF-Assgn-41

def find\_ten\_substring(num\_str):

#Remove pass and write your logic here

m=[]

l=[]

for i in range(0,len(num\_str)):

for j in range(i,len(num\_str)):

m.append(num\_str[i:j+1])

for k in set(m):

s=0

for b in range(0,len(k)):

s=s+int(k[b])

if(s==10 and b+1==len(k)):

l.append(k)

return l

num\_str="2825302"

print("The number is:",num\_str)

result\_list=find\_ten\_substring(num\_str)

print(result\_list)

Assignment 42:

Given a number n, write a program to find the sum of the largest prime factors of each of nine consecutive numbers starting from n.  
g(n) = f(n) + f(n+1) + f(n+2) + f(n+3) + f(n+4) + f(n+5) + f(n+6) + f(n+7) + f(n+8)  
where, g(n) is the sum and f(n) is the largest prime factor of n  
  
For example,  
g(10)=f(10)+f(11)+f(12)+f(13)+f(14)+f(15)+f(16)+f(17)+f(18)  
        =5 + 11 + 3 + 13 + 7 + 5 + 2 + 17 + 3  
        =66

#PF-Assgn-42

def find\_factors(num):

#Accepts a number and returns the list of all the factors of a given number

factors = []

for i in range(2,(num+1)):

if(num%i==0):

factors.append(i)

return factors

def is\_prime(num, i):

#Accepts the number num and num/2 --> i and returns True if the number is prime ,else returns False

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):

#Accepts the list of factors and returns the largest prime factor

large=[]

for i in list\_of\_factors:

if is\_prime(i,i//2)==True:

large.append(i)

return max(large)

def find\_f(num):

#Accepts the number and returns the largest prime factor of the number

f=find\_factors(num)

l=find\_largest\_prime\_factor(f)

return l

def find\_g(num):

#Accepts the number and returns the sum of the largest prime factors of the 9 consecutive numbers starting from the given number

sum=0

consicutive=[i for i in range(num,num+9)]

for i in consicutive:

largest\_prime\_factor=find\_f(i)

sum=sum+largest\_prime\_factor

return sum

#Note: Invoke function(s) from other function(s), wherever applicable.

print(find\_g(10))

Assignment 43:

Write a python function **find\_smallest\_number()** which accepts a number n and returns the smallest number having n divisors.  
Handle the possible errors in the code written inside the function.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| 16 | 120 |

#PF-Assgn-43

def find\_factors(num):

#Accepts a number and returns the list of all the factors of a given number

factors = []

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

if(num%i==0):

factors.append(i)

return factors

def find\_smallest\_number(num):

#start writing your code here

i=int(1)

while(True):

x=find\_factors(i)

if(len(x)==num):

print(x)

break

else:

i=i+int(1)

return x[-1]

num=16

print("The number of divisors :",num)

result=find\_smallest\_number(num)

print("The smallest number having",num," divisors:",result)

Assignment 44:

Write a python function **find\_duplicates()**, which accepts a list of numbers and returns another list containing all the duplicate values in the input list. If there are no duplicate values, it should return an empty list.  
Also write the pytest test cases to test the program.

|  |  |
| --- | --- |
| **Sample Input** | **Expected Output** |
| [12,54,68,759,24,15,12,68,987,758,25,69] | [12, 68] |

#PF-Assgn-44

def find\_duplicates(list\_of\_numbers):

#start writing your code here

dup=[]

for x in list\_of\_numbers:

i=list\_of\_numbers.index(x)

for a in range(i+1,len(list\_of\_numbers)):

if x==list\_of\_numbers[a]:

if x in dup:

continue

else:

dup.append(x)

return dup

list\_of\_numbers=[1,2,2,3,3,3,4,4,4,4]

list\_of\_duplicates=find\_duplicates(list\_of\_numbers)

print(list\_of\_duplicates)

Assignment 45:

The below function is written to check whether a given three digit number is an Armstrong number.  
  
**Hint**: An “Armstrong number” is an n-digit number that is equal to the sum of the nth powers of its individual digits.  
Example: 371 is an Armstrong number as 371 = 3^3 +7^3+ 1^3  
  
But the function is having errors/bugs, debug the program using the Eclipse debugger and correct it.

#PF-Tryout

def find\_armstrong(number):

temp=0

while(number!=0):

remainder=number%10

number=number/10

temp+=(remainder\*remainder\*remainder)

if(number==temp):

return True

return False

number=371

if(find\_armstrong(number)):

print(number,"is an Armstrong number")

else:

print(number,"is not an Armstrong number")