Problem Statement

Procedure to generate multiplication tables.

In [9]:

```
def mul_table(n):
    i=1
    while i<=10:
        print(f"{n}*{i}=",n*i)
        i+=1
```

In [11]:

```
mul_table(10)
```

```
10*1= 10
10*2= 20
10*3= 30
10*4= 40
10*5= 50
10*6= 60
10*7= 70
10*8= 80
10*9= 90
10*10= 100
```

Problem Statement

Procedure to return the list of factors of a given number.

Constraints

Test Cases

factorsList(6) -> [1, 2, 3, 6] factorsList(100) -> [1, 2, 4, 5, 10, 20, 25, 50, 100] factorsList(1) -> [1]

In [12]:

```
def factor_list(n):
    i=1
    while i<=n:
        if n%i==0:
            print(i)
        i+=1
```

In [14]:

```
factor_list(100)
```

```
1
2
4
5
10
20
25
50
100
```

Problem Statement

Design a procedure to determine if a given string is a Palindrome

Constraints

Test Cases

Palindrome("racecar") -> True Palindrome("python") -> False

In [5]:

```
string="racecar"
print(string[6])
```

```
r
```

In [30]:

```
def palindrome(string):
    l=len(string)
    i=l-1
    rev_string=''
    while i>=0:
        rev_string=rev_string+string[i]
        i-=1
    if (rev_string==string):
        return True
    else:
        return False
```

In [31]:

```
palindrome("python")
```

Out[31]:

```
False
```

In [32]:

```
palindrome("racecar")
```

Out[32]:

```
True
```

In [33]:

```
palindrome("sandhya")
```

Out[33]:

```
False
```

In [34]:

```
palindrome("dad")
```

Out[34]:

```
True
```

Power of a number using Iteration

In [35]:

```
def powerOfNum(base,exponent):
    i=0
    power=1
    while i<exponent:
        power=power*base
        i+=1
    print(power)
```

In [36]:

```
powerOfNum(5,2)
```

```
25
```

In [37]:

```
powerOfNum(5,3)
```

```
125
```

Power of a number using recursion

In [38]:

```
def powerOfNumUsingRecursion(base,exponent):
    if base==1:
        return 1
    elif exponent==1:
        return base
    else:
        return base*powerOfNumUsingRecursion(base,exponent-1)
```

In [39]:

```
powerOfNumUsingRecursion(5,2)
```

Out[39]:

```
25
```

In [40]:

```
powerOfNumUsingRecursion(4,3)
```

Out[40]:

```
64
```

In [41]:

```
powerOfNumUsingRecursion(1,3)
```

Out[41]:

```
1
```

In [42]:

```
powerOfNumUsingRecursion(3,1)
```

Out[42]:

```
3
```

GCD of a given number using Iteration

In [43]:

```
def findGCD(n1,n2):
    gcd=1
    i=1
    while i<=n1 and i<=n2:
        if(n1%i==0 and n2%i==0):
            gcd=i
        i+=1
    print(f'GCD of {n1} and {n2} is {gcd}')
```

In [44]:

```
findGCD(6,7)
```

```
GCD of 6 and 7 is 1
```

In [45]:

```
findGCD(3,6)
```

```
GCD of 3 and 6 is 3
```

In [46]:

```
findGCD(5,10)
```

```
GCD of 5 and 10 is 5
```

GCD of Number using Recursion

In [47]:

```
def findGCDUsingRecursion(n1,n2):
    if n2!=0:
        return findGCDUsingRecursion(n2,n1%n2)
    else:
        return n1
```

In [48]:

```
findGCDUsingRecursion(3,6)
```

Out[48]:

```
3
```

In [49]:

```
findGCDUsingRecursion(3,0)
```

Out[49]:

```
3
```

In [50]:

```
findGCDUsingRecursion(81,9)
```

Out[50]:

```
9
```

In [52]:

```
findGCDUsingRecursion(123,9)
```

Out[52]:

```
3
```

In [ ]:

```

```