PRODDEC PYTHON DAY 6

Recursive Function Explanation.

In [2]:
```
def fact(num):
    if num==1:
        return 1
    else:
        print("current number: ",num)
        return(num*fact(num-1))

num=int(input("Enter a number to get the factorial: "))
print(fact(num))
```

```
Enter a number to get the factorial: 6
current number:  6
current number:  5
current number:  4
current number:  3
current number:  2
720
```

Webbrowser module

In [12]:
```
"""
Webbrowser module is used to open a particular
webpage using our python program.
It opens webpage in our default web browser.
"""
import webbrowser as wb # wb becomes an alias for webbrowser module
wb.open("https://www.instagram.com/proddec")
```

Out[12]:
```
True
```

time.time() example in real scenario.

In [17]:
```
import time
sum1=sum2=0  # Variable for storing sum.

# Find sum using the generator.
t1=time.time()
# Creating a generator using generator comrehension.
gen=(x for x in range(1,10000000))
for i in gen:                  # Iterating through generator.
    sum1+=i                    # Finding sum of numbers from 1 to 9999999.
t2=time.time()
print("Sum using generator: ",sum1)


# Find sum using the list comprehension.
t3=time.time()
number_list=[x for x in range(1,10000000)] # List comprehension.
for i in number_list:              # Iterating through list
    sum2+=i                        # Finding sum of numbers from 1 to 9999999
t4=time.time()
print("Sum using list: ",sum2)

# Time required to run generator code
print("Time taken by generator: ",t2-t1,"sec")

# Time required to run list code
print("Time taken by list: ",t4-t3,"sec")
# We can see that even though the results are same in both the operations.
# The time taken is different.
# A programmer always uses a code that runs faster.
```

```
Sum using generator:  49999995000000
Sum using list:  49999995000000
Time taken by generator:  2.106792449951172 sec
Time taken by list:  2.2437362670898438 sec
```

List comprehension example

In [19]:
```
# Without list comprehension

list1=[1,2,3,4,5,6,7,8]


# Suppose we need to create another list from list1 such that
# The new list contains only elements greater than 5.

new_list=[]              # Defining an empty list
for element in list1:
    if element>5:
        new_list.append(element)
print(new_list)
```

```
[6, 7, 8]
```

In [20]:
```
# Using list comprehension
list1=[1,2,3,4,5,6,7,8]

new_list=[element for element in list1 if element>5]
print(new_list)


# We can see that the results are exactly the same.
# However we have written significantly lesser code.
# This is an advantage of using list comprehension in python.
```

```
[6, 7, 8]
```

In [21]:
```
# Another example of list comprehension
string1="proddec"
lst=[char for char in string1]
print(lst)
```

```
['p', 'r', 'o', 'd', 'd', 'e', 'c']
```

Nested if else example

In [22]:
```
age = int(input('Enter your age: '))
if age > 21:
    if age > 65:
        print('You are overaged for the job')
    else:
        print('Welcome, you are of the right age!')
else:
    print('You are too young for this job!')
```

```
Enter your age: 44
Welcome, you are of the right age!
```

More on tuples

In [23]:
```
# To create a tuple
tup1=(1,2,3,4,5,6)

# We can access each element using indexing.
print(tup1[3])

# However tuples are immutable.
# ie, tup1[4]=20 is an invalid command

# We can use slicing operation on tuples
print(tup1[:3])

# We can add 2 tuples
tup2=(7,8,9)
tup3=tup1 + tup2
print(tup3)
```

```
4
(1, 2, 3)
(1, 2, 3, 4, 5, 6, 7, 8, 9)
```

Frequently used modules in python

In [ ]:
```
# math
# os
# sys
# time
```