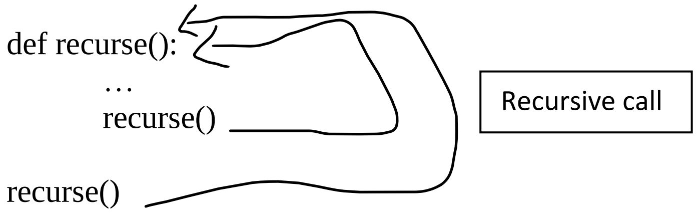
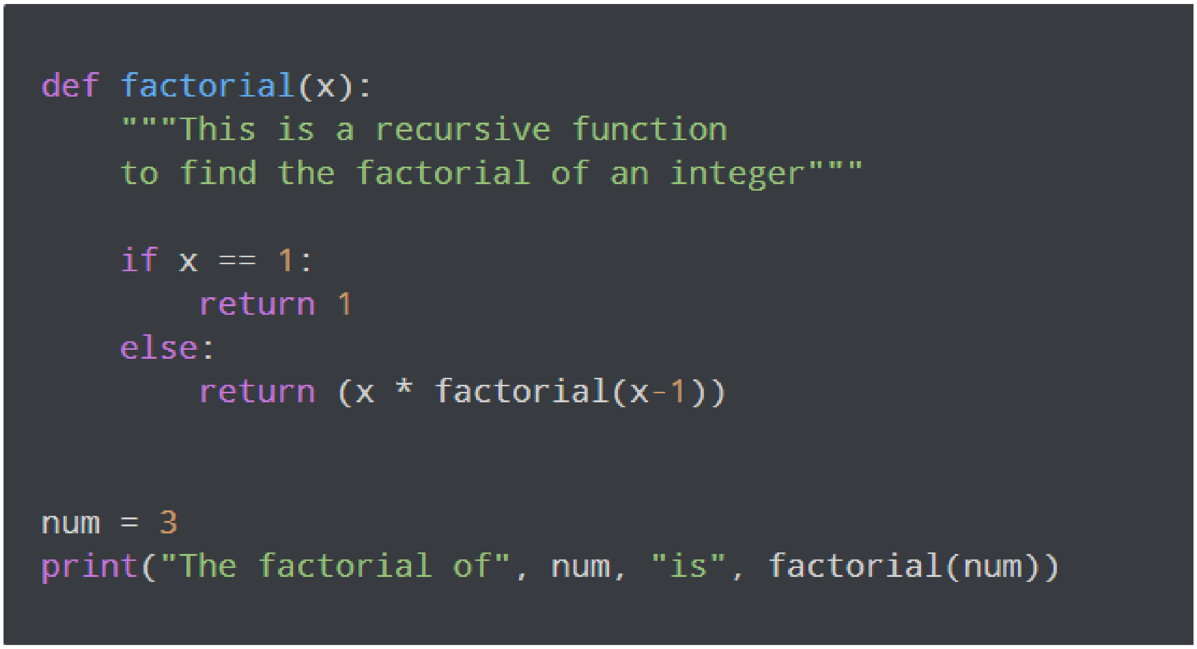
Function Recursion:

1. Recursion is the process of defining something in terms of itself.
2. When a function calls itself, it is known as recursion.
3. A physical world example would be to place two parallel mirrors facing each other. Any object in between them would be reflected recursively.



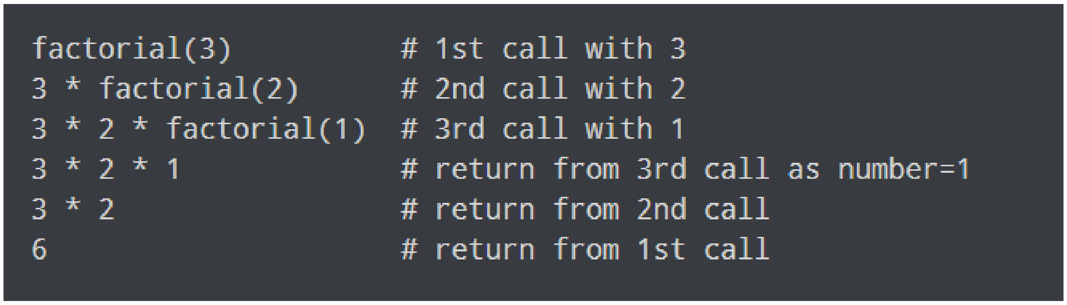
```
def recurse():
        …
        recurse()

recurse()
```

Example of recursive function (Program of Factorial):



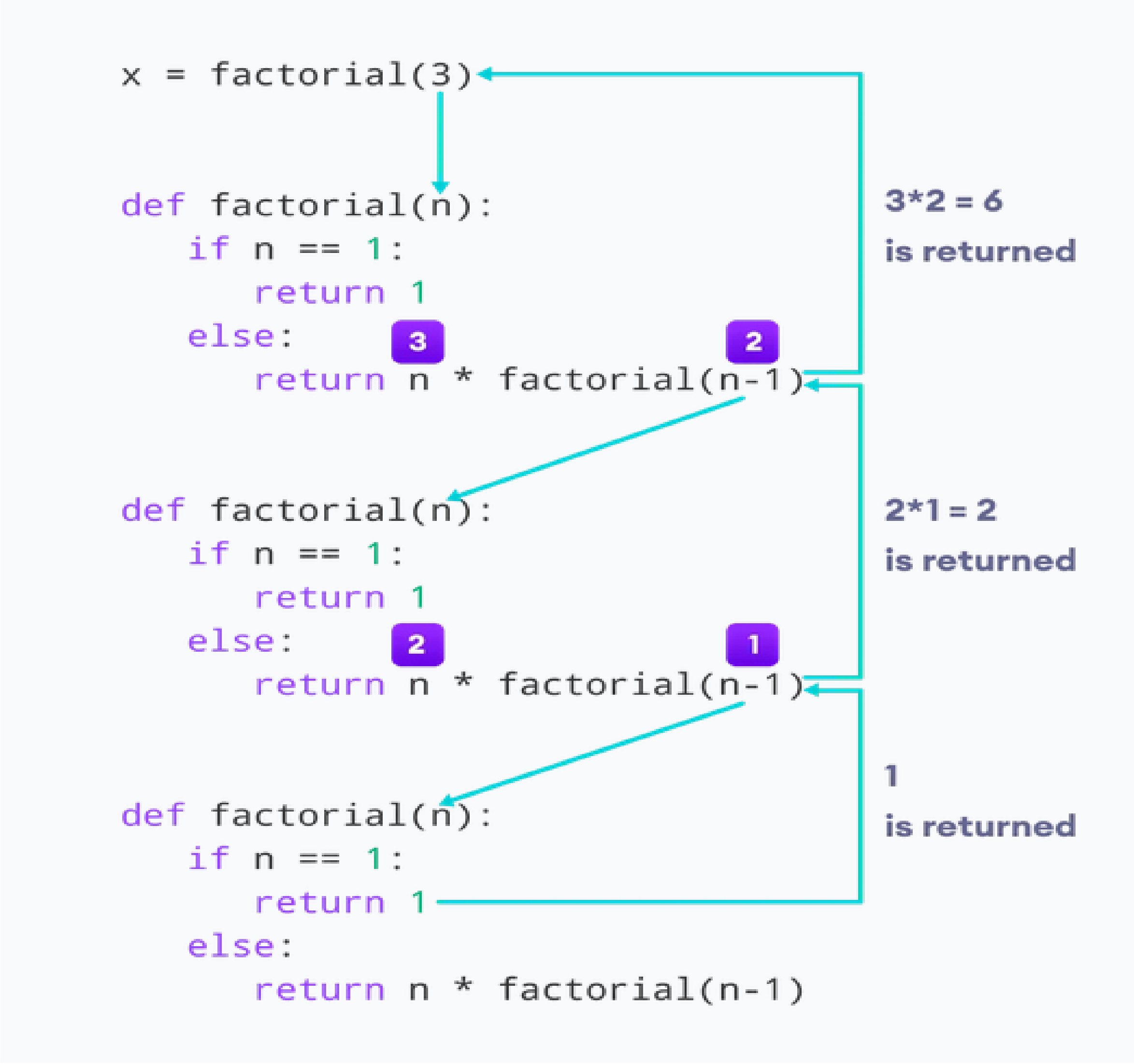
```
def factorial(x):
    """This is a recursive function
    to find the factorial of an integer"""

    if x == 1:
        return 1
    else:
        return (x * factorial(x-1))


num = 3
print("The factorial of", num, "is", factorial(num))
```

Recursive call:



```
factorial(3)          # 1st call with 3
3 * factorial(2)      # 2nd call with 2
3 * 2 * factorial(1)  # 3rd call with 1
3 * 2 * 1             # return from 3rd call as number=1
3 * 2                 # return from 2nd call
6                     # return from 1st call
```

Working:



```
x = factorial(3)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)
```

Advantages:

1. Recursive functions make the code look clean and elegant.
2. A complex task can be broken down into simpler sub-problems using recursion.
3. Sequence generation is easier with recursion than using some nested iteration.

Disadvantages:

1. Sometimes the logic behind recursion is hard to follow through.
2. Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
3. Recursive functions are hard to debug.

Tail Recursion:

1. A unique type of recursion where the last procedure of a function is a recursive call.
2. The recursion may be automated away by performing the request in the current stack frame and returning the output instead of generating a new stack frame.
3. The tail-recursion may be optimized by the compiler which makes it better than non-tail recursive functions.

1. Explain the step by step working of this code and predict the output:

```
def recursive_something(n):
  if n <= 1:
     return n
  else:
     return(recursive_something(n-1) + recursive_something(n-2))
n_terms = 10
# check if the number of terms is valid
if n_terms <= 0:
  print("Invalid input ! Please input a positive value")
else:
  print("This series is ____")
  for i in range(n_terms):
     print(recursive_something(i))
```

Solution:

```
This series is ____
0
1
1
2
3
5
8
13
21
34
```

2. Explain the step by step working of this code and predict the output:

```
def recursion(k):
 if(k > 0):
  result = k + recursion(k - 1)
  print(result)
 else:
  result = 0
 return result
print("Recursion Example Results")
recursion(3)
```

Solution:

```
Recursion Example Results
1
3
6
```

3. Explain the step by step working of this code and predict the output:

```
total = 0
def sum_nestedlist( l ):
  global total
  for j in range(len(l)):
    if type(l[j]) == list :
      sum_nestedlist(l[j])
    else:
      total += l[j]
sum_nestedlist([[1,2,3],[4,[5,6]],7])
print(total)
```

Solution:

```
 28
```

4. Predict the output of the program:

```
def printPattern(targetNumber) :
  if (targetNumber <= 0) :
  print(targetNumber)
```

```
    return

  print(targetNumber)
  printPattern(targetNumber - 5)
  print(targetNumber)

n = 10
printPattern(n)
```

Solution:

```
10
5
0
5
10
```

5. Explain the step by step working of this code and predict the output:

```
def P(n, x):
    if(n == 0):
        return 1
    elif(n == 1):
        return x
    else:
        return (P(n-1, x)+(n-1)*P(n-2, x))
n = 3
X = 5
print(P(n, X))
```

Solution:

16

6. Explain the step by step working of this code and predict the output [Example of tail recursion].

```
def Recur_facto(n, a = 1):
    if (n == 0):
        return a
    return Recur_facto(n - 1, n * a)

# print the result
print(Recur_facto(6))
```

Solution:
720

7. Explain the step by step working of this code and predict the output:

```
def pascal(n):
   if n == 1:
      return [1]
   else:
      line = [1]
      previous_line = pascal(n-1)
      for i in range(len(previous_line)-1):
         line.append(previous_line[i] + previous_line[i+1])
      line += [1]
   return line
print(pascal(6))
```

Solution:
[1, 5, 10, 10, 5, 1]

8. Predict the output of the function:

```
houses = ["Eric's house", "Kenny's house", "Kyle's house", "Stan's house"]
def deliver_presents_recursively(houses):
   if len(houses) == 1:
      house = houses[0]
      print("Delivering presents to", house)

   else:
      mid = len(houses) // 2
      first_half = houses[:mid]
      second_half = houses[mid:]

      deliver_presents_recursively(first_half)
      deliver_presents_recursively(second_half)
deliver_presents_recursively(houses)
```

Solution:
Delivering presents to Eric's house
Delivering presents to Kenny's house
Delivering presents to Kyle's house
Delivering presents to Stan's house

9. Explain the step by step working of this code and predict the output:

```
def mult3(n):
   if n == 1:
      return 3
   else:
      return mult3(n-1) + 3

for i in range(1,10):
   print(mult3(i))
```

Sol.

```
3
6
9
12
15
18
21
24
27
```

10. Explain the step by step working of this code and predict the output:

```
current_number = 1
accumulated_sum = 0
def sum_recursive():
   global current_number
   global accumulated_sum
   # Base case
   if current_number == 11:
      return accumulated_sum
   # Recursive case
   else:
      accumulated_sum = accumulated_sum + current_number
      current_number = current_number + 1
      return sum_recursive()
a= sum_recursive()
print(a)
```

Solution:

55

11. Explain the step by step working of this code and predict the output:

```
def printSubsequences(arr, index, subarr):
   if index == len(arr):
      if len(subarr) != 0:
            print(subarr)
        else:
            printSubsequences(arr, index + 1, subarr)

            printSubsequences(arr, index + 1,
                                subarr+[arr[index]])
        return
    arr = [1, 2, 3]
    printSubsequences(arr, 0, [])
```

Solution:

```
[3]
[2]
[2, 3]
[1]
[1, 3]
[1, 2]
[1, 2, 3]
```

12. Explain the step by step working of this code and predict the output:

```
def power(N, P):

    if(P == 0 or P == 1):

        return N

    else:

        return (N*power(N, P-1))

N = 5

P = 2

print(power(N, P))
```

Solution:

25

13. Explain the step by step working of this code and predict the output.

```
def findnum(Arr,n):
    if n == 1:
        return Arr[0]
    else:
        return min(Arr[n-1],findnum(Arr,n-1))

A = [1, 4, 24, 17, -5, 10, -22]
n = len(A)
```

```
print(findnum(A,n))
```

Solution:
-22

14. Explain the step by step working of this code and predict the output.

```
def remove(string):
  if not string:
    return ""

  if string[0] == "\t" or string[0] == " ":
    return remove(string[1:])
  else:
    return string[0] + remove(string[1:])

print(remove("This is the tutorial of Python"))
```

Solution:
ThisisthetutorialofPython