Recursion

- A recursive function is a function that calls itself, where successive calls reduce a computation to smaller computations of the same type until a base case with a trivial solution is reached.

    - **Example**

```
def power(r, n):
## iterative definition of power function
value = 1
for i in range(1, n + 1):
      value = r * value
      return value

print(power(2, 3))
```

Output

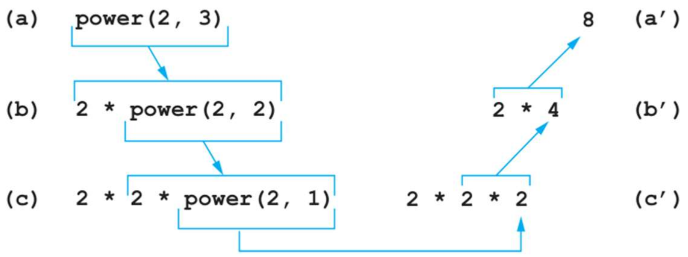
```
8
```

    - **Example**

```
def power(r, n):
## recursive definition of power function
if n == 1:
      return r
else:
      return r * power(r, n - 1)
print(power(2, 3))
```

Output

```
8
```



- Recursive algorithms have two traits.
    - There are one or more **base cases** with trivial solutions.
    - There is an "**inductive step**" that successively reduces the problem to smaller versions of the same problem, with the reduction eventually culminating in a base case. This inductive step is called the reducing step.
- **Pseudocode of recursion**

```
if a base case is reached
      Solve the base case directly.
else
      Repeatedly reduce the problem to a version increasingly
      closer to a base case until it becomes a base case.
```

- **Advantages of Recursion**
  - Recursive functions make the code look clean and elegant.
  - A complex task can be broken down into simpler sub-problems using recursion.
  - Sequence generation is easier with recursion than using some nested iteration.
  - We can reduce the length of code and become more readable and understandable to the user/ programmer.
- **Disadvantages of Recursion**
  - Sometimes the logic behind recursion is hard to follow through.
  - Recursive calls are expensive (inefficient) as they take up a lot of memory and time.
  - Recursive functions are hard to debug.
  - Recursion can lead to stack overflow errors if the recursion depth is too high.