In [1]:
```
def recur(n):
    if n < 0: # base condition
        return

    print(n)
    recur(n-1)

recur(10)
```

```
10
9
8
7
6
5
4
3
2
1
0
```

In [3]:
```
def recur(n, i=0):
    if i > n: # base condition
        return

    print(i)
    recur(n, i+1)

recur(10)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [6]:
```
def _iter(n):
    i = 0
    while i <= n:
        print(i)
        i+=1

_iter(10)
```

```
0
1
2
3
4
5
6
7
8
9
10
```

In [ ]:
```

```

In [ ]:
```
### Fibonacci series
0 1 1 2 3 5  8 ....
1 2 3 4 5 6  7 .... nth
```

In [9]:
```
def fib(n):
    if n == 1:
        return 0
    if n == 2:
        return 1

    return fib(n-1) + fib(n-2)

print(fib(4))
print(fib(5))
print(fib(6))
print(fib(7))
print(fib(50))
```

```
2
3
5
8

---------------------------------------------------------------------------
KeyboardInterrupt                         Traceback (most recent call last)
Cell In[9], line 13
     11 print(fib(6))
     12 print(fib(7))
---> 13 print(fib(50))

Cell In[9], line 7, in fib(n)
      4 if n == 2:
      5     return 1
----> 7 return fib(n-1) + fib(n-2)

Cell In[9], line 7, in fib(n)
      4 if n == 2:
      5     return 1
----> 7 return fib(n-1) + fib(n-2)

    [... skipping similar frames: fib at line 7 (30 times)]

Cell In[9], line 7, in fib(n)
      4 if n == 2:
      5     return 1
----> 7 return fib(n-1) + fib(n-2)

Cell In[9], line 2, in fib(n)
      1 def fib(n):
----> 2     if n == 1:
      3         return 0
      4     if n == 2:

KeyboardInterrupt:
```

In [10]:
```
# top to bottom
cache = {} # unorderd_map<int, int>

def fib(n):
    if n == 1:
        return 0
    if n == 2:
        return 1

    if n in cache:
        return cache[n]

    r = fib(n-1) + fib(n-2)
    cache[n] = r
    return r

print(fib(4))
print(fib(5))
print(fib(6))
print(fib(7))
print(fib(50)) # linear
```

```
2
3
5
8
7778742049
```

In [14]:
```
# memoization
cache = {1:0, 2:1} # unorderd_map<int, int>
N

cachhe = [n .... -1]
def fib(n):
    if n <= 0:
        raise Exception("n should be >= 1")

    if n in cache:
        return cache[n]

    r = fib(n-1) + fib(n-2) # recurrance relation
    cache[n] = r
    return r

print(fib(4))
print(fib(5))
print(fib(6))
print(fib(7))
print(fib(50)) # linear
print(fib(0))

# TC: O(n)
# SC: O(n)
```

```
2
3
5
8
7778742049

---------------------------------------------------------------------------
Exception                                 Traceback (most recent call last)
Cell In[14], line 19
     17 print(fib(7))
     18 print(fib(50)) # linear
---> 19 print(fib(0))

Cell In[14], line 5, in fib(n)
      3 def fib(n):
      4     if n <= 0:
----> 5         raise Exception("n should be >= 1")
      7     if n in cache:
      8         return cache[n]

Exception: n should be >= 1
```

In [ ]:
```

```

In [19]:
```python
# bottom to top
# tabulation
results = [0,1]
def fib(n):
    while len(results) < n:
        curr = results[-1] + results[-2] # results[len(results) - 1] + results
        results.append(curr)

    return results[n-1]

print(fib(1))
print(fib(2))
print(fib(3))
print(fib(4))
print(fib(5))
print(fib(6))

# TC: O(N)
# SC: O(N)
```

```
0
1
1
2
3
5
```

In [20]:
```python
def fib(n):
    a = 0
    b = 0
    c = 1
    while n > 0:
        a = b
        b = c
        c = a + b
        n -= 1

    return a

print(fib(1))
print(fib(2))
print(fib(3))
print(fib(4))
print(fib(5))
print(fib(6))

# TC: O(n)
# SC: O(1)
```

```
0
1
1
2
3
5
```

In [ ]:

Question: 1D
https://leetcode.com/problems/climbing-stairs/ (https://leetcode.com/problems/climbing-stairs/)

prerna

```
class Solution {
public:
    unordered_map<int,int> mp;
    int climbStairs(int n) {
        if(n==0) return 0;
        if(n==1) return 1;
        if(n==2) return 2;
        if(mp.count(n)>0) return mp[n];
        int r = climbStairs(n-1)+climbStairs(n-2);
        mp[n] = r;
        return r;
    }
};
```

Rajat Kumar

```
public int rec(int n ,Integer dp[]){
        if(n==0) return 1;
        if(n<0) return 0;
        if(dp[n] != null) return dp[n];
        return dp[n] = rec(n-1,dp) + rec(n-2,dp);
    }
```

valeti

```
class Solution {
public:
    std::map<int,int> cacheMap;

    int climbUtil(int n) {

        if (cacheMap.find(n) != cacheMap.end()) {
            return cacheMap.at(n);
        }
```

```

```

```
def climbStairs(n):
    if n <= 0:
        return 0
    elif n == 1:
        return 1
    elif n == 2:
        return 2

    prev_1 = 1
    prev_2 = 2

    for _ in range(3, n + 1):
        curr = prev_1 + prev_2
        prev_1, prev_2 = prev_2, curr

    return prev_2
```

```java
class Solution {
    public int climbStairs(int n) {
        int f1,f2,f3=0,i;
        if(n==1)
            return 1;
        else if(n==2)
            return 2;
        else
        {
            f1=1;
            f2=2;
            for(i=2;i<n;i++)
            {
                f3=f2+f1;
                f1=f2;
                f2=f3;
            }

        }
        return f3;
    }
}
```

TC: O(N)
SC: O(1)

```java
class Solution {
public:
    int climbStairs(int n) {
       // 1-> 1
       // 2-> 2
       // n -> n-1 + n-2

       int a = 1;
       int b = 2;
       int curr = 0;
       while (n > 0) {
           curr = a;
           a = b;
           b = curr + a;
           n -= 1;
       }

       return curr;
    }

    // 1->1
    // 2->
};
```

```
Class Solution {
public:
    int climbStairs(int n) {
        if(n==0) return 0;
        if(n==1) return 1;
        if(n==2) return 2;
        int p1= 1;
        int p2= 2;
        for(int i=3;i<n+1;i++)
        {
            int curr = p1 + p2;
            p1=p2;
            p2 = curr;
        }
```

In [ ]:

In [ ]:

In [ ]:

Question

https://leetcode.com/problems/unique-paths/ (https://leetcode.com/problems/unique-paths/)

Iterative Optimized

```
class Solution:
    def uniquePaths(self, m: int, n: int) -> int:
        if m == 1 and n == 1:
            return 1

        pathList = [1 for i in range(n)]

        for i in range(1, m):
            for j in range(n):
                if j > 0:
                    pathList[j] = pathList[j] + pathList[j - 1]

        return pathList[-1]
```

```
TC: O(m*n)
SC: O(n)
```

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<vector<int>> nestedVec(m, vector<int>(n,1));
        for (int i=1; i<m; i++) {
            for (int j=1; j<n; j++) {
                nestedVec[i][j] = nestedVec[i][j-1] + nestedVec[i-1]
[j];
            }
        }
        return nestedVec[m-1][n-1];
    }
};
TC: O(m*n)
SC: O(m*n)
```

```
public int uniquePaths(int m, int n) {

        int row[] = new int[n];
        Arrays.fill(row, 1);

        for(int i = 1; i < m; i++){
            for(int j = 1; j < n; j++){
                row[j] = row[j-1] + row[j];
            }
        }
        return row[n - 1];
    }
```

In [ ]:

Top Down: Using matrix cache

```
class Solution {
public:
    int uniquePaths(int m, int n) {
        vector<vector<int>> nestedVec(m, vector<int>(n,-1));
        for (int i = 0; i < m; i++) {
            nestedVec[i][0] = 1;
        }
        for (int i = 0; i < n; i++) {
            nestedVec[0][i] = 1;
        }

        return uniquePathsUtil(m,n, cache);
    }
```

Top Down: Using hashmap cache. Assuming first cell is 1,1 instead of 0,0

```
typedef unordered_map<pair<int, int>, int> cacheType;
class Solution {
public:
    int uniquePaths(int m, int n) {
        cacheType cache;

        cache[make_pair<int,int>(1,1)] = 1;


        return uniquePathsUtil(m,n, cache);
    }

    int uniquePathsUtil(int m, int n, cacheType& cache) {
        if (m <= 0 || n <= 0) return 0;

        if (cache.count(make_pair<int,int>(m,n)) != 0)
            return cache(make_pair<int,int>(m,n));

        int res  = uniquePathsUtil(m-1, n, cache) + uniquePathsUtil
(m, n-1, cache);
        cache(make_pair<int,int>(m,n)) = res;
        return res;
    }
};
```

In [ ]:
```
[a, b ,c , d]

(ab)
(ba)
```

In [25]:
```
def longest_char_seq(s):
    max_count = 1
    c = s[0]
    count = 1
    for i in range(1, len(s)):
        if c == s[i]:
            count += 1
            max_count = max(max_count, count)
        else:
            c = s[i]
            count = 1

    return max_count

longest_char_seq('abaaaabaaaaaaaabbbbbbbbbbbbbbb')


['abbbbbbbba', 'aaba', 'aaabba']
```

Out[25]: 15

https://app.codility.com/programmers/task/concatenating_of_words/
(https://app.codility.com/programmers/task/concatenating_of_words/)