

In [2]:

```
# 실습을 위해 반드시 먼저 실행해주세요.  
from IPython.display import IFrame
```

재귀 함수(recursive function)

재귀 함수란 반복(루프)을 위해 직접 또는 간접적으로 자기 자신을 호출하는 함수를 뜻한다.

재귀는 간단한 반복을 대신할 수도 있지만, 반드시 더 간단하거나 효율적인 것은 아니다.

In []:

```
# 반복문  
n = [1, 2, 3, 4, 5]  
sum = 0  
for i in n:  
    sum += i  
sum
```

In [3]:

```
# 재귀 함수  
def my_sum(n):  
    print(n)  
    if not n:  
        return 0  
    else:  
        return n[0] + my_sum(n[1:])
```

In [4]:

```
my_sum([1, 2, 3, 4, 5])
```

```
[1, 2, 3, 4, 5]  
[2, 3, 4, 5]  
[3, 4, 5]  
[4, 5]  
[5]  
[]
```

Out[4]:

15

실습문제 - 팩토리얼 계산 (반복문)

1 부터 n 까지 양의 정수를 차례대로 곱한 값

팩토리얼 (factorial) 을 계산하는 함수 `fact(n)` 를 작성해봅시다.

n은 1보다 큰 정수라고 가정하고, 팩토리얼을 계산한 값을 반환합니다.

$$n! = \prod_{k=1}^n k$$

$$n! = 1 * 2 * 3 * \dots * (n-1) * n$$

예시 출력)

```
fact(4)
```

In []:

```
# 반복문을 통해 아래에 코드를 작성해주세요.
def fact(n):
    for i in range(1, n):
        n = n*i
    return n
```

In []:

```
def fact(n):
    result = 1
    while n > 1:
        result = result * n
        n -= 1
    return result
```

In []:

```
# fact(4) 함수를 호출해주세요.
fact(4)
```

실습 문제 - 팩토리얼 계산 (재귀)

```
1! = 1
2! = 1 * 2 = 1! * 2
3! = 1 * 2 * 3 = 2! * 3
4! = 1 * 2 * 3 * 4 = 3! * 4
```

In [18]:

```
# 재귀함수를 통해 아래에 코드를 작성해주세요.
def factorial(n):
    if n == 1:
        return 1
    else:
        return n * factorial(n-1)

# print(n)
# if not n:
#     return 1
# else:
#     return n * factorial(n-1)
```

In [19]:

```
# 아래의 코드를 통해 factorial(4) 함수를 호출해주세요.
factorial(4)
```

Out[19]:

24

In [20]:

```
IFrame("https://goo.gl/8Cmw2X", width='100%', height='500px')
```

Out[20]:

Write code in Python 3.6



Visualize Execution

Live Programming
Mode

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

[illegible]

Hey there ?

```
-----
RecursionError                                Traceback (most recent call last)
<ipython-input-21-93b0c4ca1422> in <module>
      3     print('Hey there ?')
      4     greeting()
----> 5 greeting()

<ipython-input-21-93b0c4ca1422> in greeting()
      2 def greeting():
      3     print('Hey there ?')
----> 4     greeting()
      5 greeting()

... last 1 frames repeated, from the frame below ...

<ipython-input-21-93b0c4ca1422> in greeting()
      2 def greeting():
      3     print('Hey there ?')
----> 4     greeting()
      5 greeting()

RecursionError: maximum recursion depth exceeded in comparison
```

In [22]:

```
def test_error(n):
    return test_error(n)
test_error(1000)
```

```
-----
RecursionError                                Traceback (most recent call last)
<ipython-input-22-febca585c5fe> in <module>
      1 def test_error(n):
      2     return test_error(n)
----> 3 test_error(1000)

<ipython-input-22-febca585c5fe> in test_error(n)
      1 def test_error(n):
----> 2     return test_error(n)
      3 test_error(1000)

... last 1 frames repeated, from the frame below ...

<ipython-input-22-febca585c5fe> in test_error(n)
      1 def test_error(n):
----> 2     return test_error(n)
      3 test_error(1000)

RecursionError: maximum recursion depth exceeded
```

실습문제 - 피보나치 수열

피보나치 수열은 다음과 같은 점화식이 있다.

피보나치 값을 리턴하는 두가지 방식의 코드를 모두 작성해보자.

$$F_0 = F_1 = 1$$

$$F_n = F_{n-1} + F_{n-2} \quad (n \in \{2, 3, 4, \dots\})$$

1) `fib(n)` : 재귀함수

2) `fib_loop(n)` : 반복문 활용한 함수

예시 입력)
`fib(10)`

In [37]:

```
# 아래에 재귀를 이용한 코드를 작성해주세요.
def fib(n):
    if n == 0 or n == 1:
        return 1
    else:
        return fib(n-1) + fib(n-2)

fib(10)

#     if n < 2:
#         return 1
#     else:
#         return fib(n-1) + fib(n-2)
```

Out[37]:

89

In [40]:

```
# 아래에 반복문을 이용한 코드를 작성해주세요.
def fib_loop(n):
    result = [1, 1]
    for i in range(1, n):
        result.append(result[-2] + result[-1])
    return result[-1]

#     numbers = [1, 1]
#     for i in range(0, n-1):
#         i = numbers[i] + numbers[i+1]
#         numbers.append(i)
#     return i
```

In [42]:

```
def fib_loop2(n):
    a, b = 1, 1
    for i in range(2, n+1):
        a, b = b, a+b
    return b

fib_loop2(10)
```

Out[42]:

89

In []:

```
# def fib_loop_while(n):
#     numbers = [1, 1]
#     while n > 1:
#         number = number[]
#         n -= 1
#     return fib_loop_while(n-1) + fib_loop_while(n-2)

# fib_loop_while(10)
```

In [33]:

```
# fib_loop(10) 함수를 호출해주세요.
fib_loop(10)
```

Out[33]:

89

반복문과 재귀 함수의 차이

- 단순한 합산을 구하기 위해 재귀를 사용하는 것은 조금 지나친 방법일 수 있다.
- 파이썬에서는 이보다 for, while 이 종종 더 현실적인 방법을 보여주며, 굳이 재귀 호출을 위한 함수를 필요로 하지 않는다.
- 특히 for 문은 대부분의 상황에서 재귀를 대신 할 수 있으며, 메모리 사용량과 실행 시간에서 더 효율적이다.
- 하지만, 재귀는 알고리즘을 구현할 때 매우 유용한 경우가 많다. (알고리즘을 구현하면 반복문보다 재귀로 구현한 코드가 더 직관 적일 때가 많다.)

두 코드 모두 원리는 같다!

1. 반복문 코드:

- n이 1보다 큰 경우 반복문을 돌며, n은 1씩 감소한다.
- 마지막에 n이 1이면 더 이상 반복문을 돌지 않는다.

1. 재귀 함수 코드:

- 재귀 함수를 호출하며, n은 1씩 감소한다.
- 마지막에 n이 1이면 더 이상 추가 함수를 호출을 하지 않는다.

- 재귀 함수는 기본적으로 같은 문제이지만 점점 범위가 줄어드는 문제를 풀게 된다.
- 재귀함수를 작성시에는 반드시, base case 가 존재 하여야 한다.
- base case 는 점점 범위가 줄어들어 반복되지 않는 최종적으로 도달하는 곳이다.

재귀를 이용한 팩토리얼 계산에서의 base case는 n이 1일때, 함수가 아닌 정수 반환하는 것이다.

In [43]:

```
# 큰 숫자를 재귀로 짜여진 fib() 함수의 인자로 넘겨보세요. (재귀)
fib(33)
```

Out[43]:

5702887

In [44]:

```
# 100배 되는 숫자를 반복문으로 짜여진 fib_loop() 인자로 넘겨보세요. (반복문)
fib_loop(3300)
```

Out[44]:

```
330153163507162264637094778670152653434758914922281728912670042596222213549775330156165336158736310
353027241745676035599689641466986559284807184964107170097095641039922133213208696287348034606696631
279857018624076816437080868866048583598564218972623531157813672221890203506955836803227784343694838
806290480685283349217035498351102885889468646619750569482644246863804467015344937199892515242806415
817865329230171700334166247742099197950515141020278273960524418471603108466460833211102223560755434
212805159313788635942586599452884874773918260022865994184698398238432381390369504872697698637028874
958687841091743740983161275336114608885705665822704734020694899622487801
```

In [45]:

```
IFrame('https://goo.gl/JZ7s15', width='100%', height='500px')
```

Out[45]:

Write code in Python 3.6

Visualize Execution

Live Programming
Mode

hide exited frames [default]

inline primitives but don't nest objects [default]

draw pointers as arrows [default]

