

재귀 함수

주요 내용

- 재귀 개념
- 재귀 함수

재귀 함수

- 재귀_{recursion}: 주어진 문제를 해결하기 위해 보다 간단한 문제들로 쪼개어 해결하는 과정을 반복하는 기법
- 분할 정복 기법으로 풀릴 수 있는 문제가 재귀를 활용하여 쉽게 해결될 수 있음
- **재귀 함수**_{recursive function}: 재귀를 이용하여 정의된 함수

예제: 리스트 항목들의 합

- 항목들의 누적합을 0으로 지정하고 시작
- 각 항목을 확인할 때마다 누적합 계산

```
In [1]: def sum(num_list):  
        # 누적 합 저장  
        the_sum = 0  
  
        # 모든 항목을 누적합에 더하기  
        for item in num_list:  
            the_sum = the_sum + item  
  
        return the_sum
```

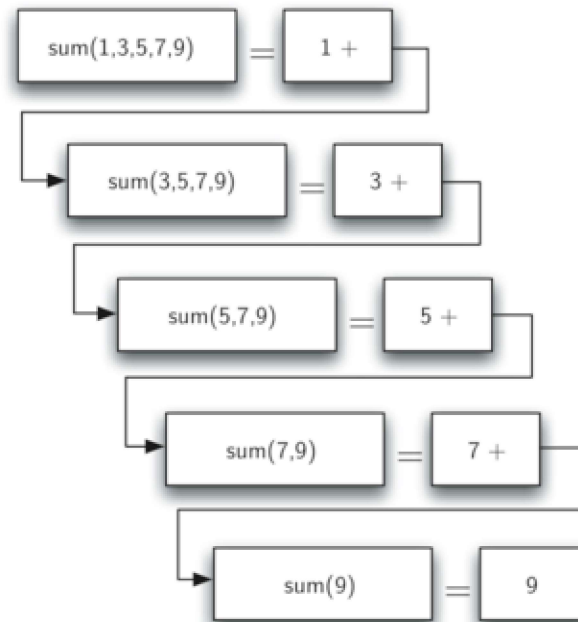
```
In [2]: print(sum([1, 3, 5, 7, 9]))
```

`the_sum` 변수가 업데이트 되는 과정:

$$((((0 + 1) + 3) + 5) + 7) + 9$$

재귀 기법 적용: `the_sum`의 값이 업데이트 되는 과정

$$1 + (3 + (5 + (7 + (9))))$$



리스트의 머리와 꼬리 활용

머리와 꼬리 개념을 이용하여 재귀를 설명하면 다음과 같다.

- **머리(head)**: 리스트의 0번 인덱스 값, 즉 `num_list[0]`
- **꼬리(tail)**: 0번 인덱스를 제외한 나머지, 즉 `num_list[1:]`

```
sum(num_list) = num_list[0] + sum(num_list[1:])
```

1. 먼저 꼬리에 재귀를 적용한다.
2. 꼬리에 대한 재귀가 특정 값을 반환할 때까지 기다린 다음 머리와 합한다.

```
In [3]: def sum(num_list):  
        if len(num_list) == 1: # 항목이 1개 일때  
            return num_list[0]  
        else:                  # 항목이 2개 이상일 때: 머리와 꼬리 재귀 결과의 합  
            return num_list[0] + sum(num_list[1:])
```


재귀 호출

- 재귀 호출_{recursive call}: 함수가 실행될 때 함수 자신을 호출하는 것
- 재귀 알고리즘_{recursive algorithm}: 재귀 호출을 이용하여 구현된 알고리즘

재귀 알고리즘의 특징

1. 재귀 호출이 반드시 발생해야 한다.
2. 종료 조건_{base case}이 존재해야 한다.
 - `sum()` 함수의 경우 `len(num_list) == 1`이 종료 조건을 다룬다.
 - 종료 조건이 없는 재귀 알고리즘은 실행이 종료되지 않을 수 있다.
3. 재귀 호출에 사용되는 입력값의 크기가 줄어들어야 하며, 결국에는 종료 조건을 만족하는 상태에 다달아야 한다.
 - `sum()` 함수의 경우 `sum(num_list[1:])`에 사용된 리스트 `num_list[1:]`의 크기는 `num_list`보다 1 작다.

예제 1: 종료 조건이 없는 경우

종료 조건이 없는 재귀 호출은 실행이 종료되지 않을 수 있다.

```
def f(x):  
    return f(x-1) + 1
```

예제 2: 종료 조건에 도달하는지 여부를 알 수 없는 경우

아래 `collatz(n)` 함수는 임의의 양의 정수 `n`에 대해 짝수면 2로 나누고, 홀수면 세 배 더하기 1을 반복적으로 실행하여 언젠가 1이 나오면 멈춘다. 즉, 종료 조건은 함수의 인자가 1일 때이다.

```
In [5]: def collatz(n):  
        if n == 1:  
            print(n)  
        elif n % 2 == 0:  
            print(n, end=', ')  
            collatz(n//2)  
        else:  
            print(n, end=', ')  
            collatz(3*n + 1)
```

```
In [6]: collatz(7)
```

7, 22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1

콜라츠 함수 입력 크기와 재귀 호출 횟수

입력값이 커질 수록 재귀 호출 횟수가 반드시 많아지는 것은 아니다.

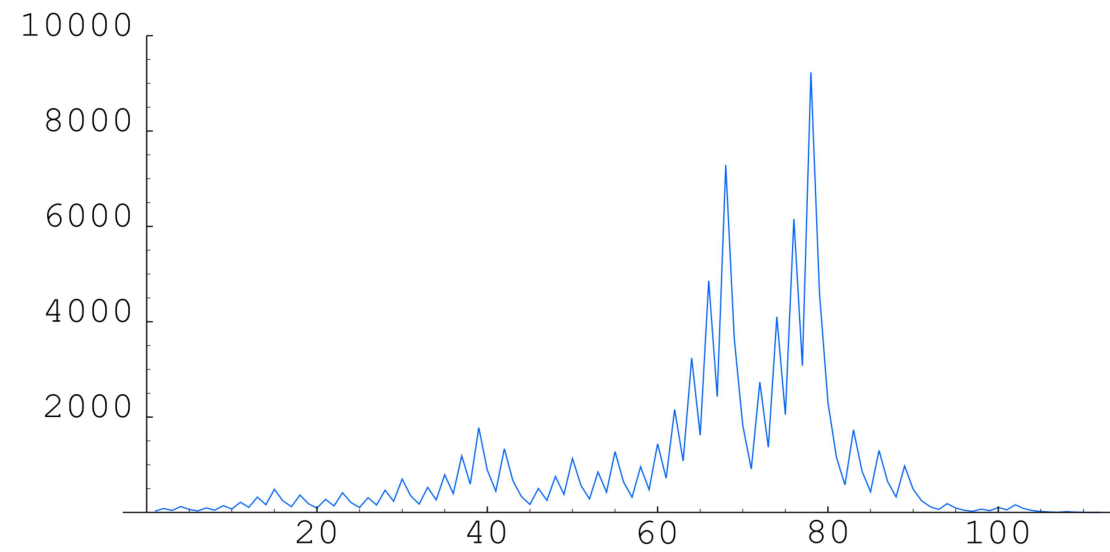
In [7]: `collatz(27)`

```
27, 82, 41, 124, 62, 31, 94, 47, 142, 71, 214, 107, 322, 161, 484, 242, 1
21, 364, 182, 91, 274, 137, 412, 206, 103, 310, 155, 466, 233, 700, 350,
175, 526, 263, 790, 395, 1186, 593, 1780, 890, 445, 1336, 668, 334, 167,
502, 251, 754, 377, 1132, 566, 283, 850, 425, 1276, 638, 319, 958, 479, 1
438, 719, 2158, 1079, 3238, 1619, 4858, 2429, 7288, 3644, 1822, 911, 273
4, 1367, 4102, 2051, 6154, 3077, 9232, 4616, 2308, 1154, 577, 1732, 866,
433, 1300, 650, 325, 976, 488, 244, 122, 61, 184, 92, 46, 23, 70, 35, 10
6, 53, 160, 80, 40, 20, 10, 5, 16, 8, 4, 2, 1
```

In [8]: `collatz(1024)`

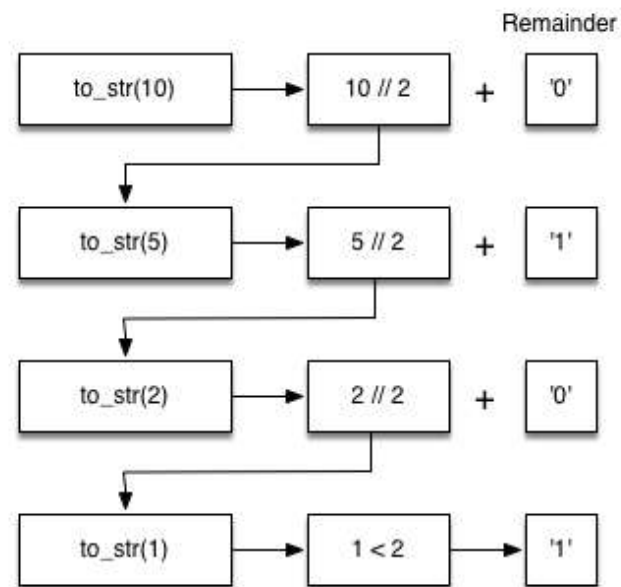
```
1024, 512, 256, 128, 64, 32, 16, 8, 4, 2, 1
```

`collatz(27)` 이 실행되는 동안 재귀 호출에 사용되는 인자의 변화:



진법 변환

아래 그림은 정수 10을 이진법으로 변환하는 과정을 보여준다.



진법 변환 함수: to_str()

```
In [9]: def to_str(n, base):  
        numeric_string = "0123456789ABCDEF"  
  
        if n < base:                # 종료 조건  
            return numeric_string[n]  
        else:                        # 재귀  
            return to_str(n // base, base) + numeric_string[n % base]
```

```
In [10]: print(to_str(46685, 2))
```

1011011001011101

```
In [11]: print(to_str(46685, 16))
```

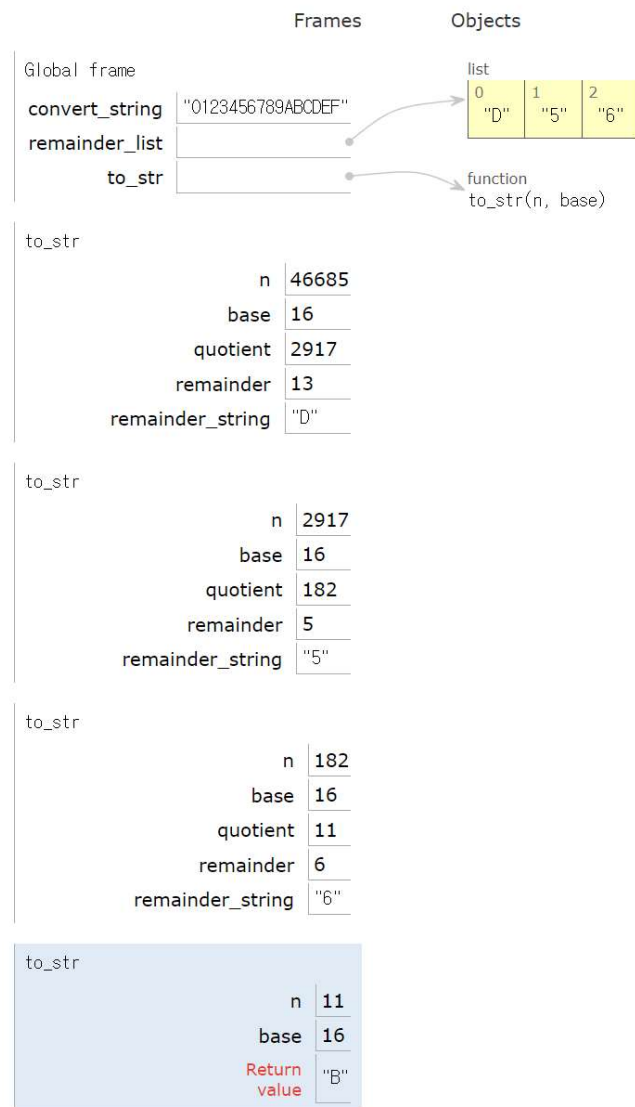
B65D

콜 스택

- 파이썬의 경우 재귀 함수가 실행되면 재귀 호출이 발생할 때마다 함수 호출의 실행을 관리하는 **프레임**_{frame} 생성
- 생성되는 프레임은 스택으로 관리되며, 이를 **콜 스택**_{call stack}이라 부름.
- 참고: [PythonTutor: 콜 스택](#)

```
In [12]: numeric_string = "0123456789ABCDEF"
         remainder_list = []

         def to_str(n, base):
             if n < base:
                 return numeric_string[n]
             else:
                 quotient = n // base
                 remainder = n % base
                 remainder_string = numeric_string[remainder]
                 remainder_list.append(remainder_string)
                 return to_str(quotient, base) + remainder_list.pop()
```



Stack Overflow

- 재귀 호출이 너무 많은 경우 스택 오버플로우로 인한 `RecursionError` 오류가 발생
- 아래 `f()` 함수는 재귀를 무한정 호출
- 파이썬은 재귀 호출에 활용되는 스택 메모리 영역을 보호하기 위해 최대 1만번 까지의 재귀 호출 허용
- 재귀 호출 허용 횟수를 지정할 수는 있지만 권장되지 않음.

```
In [15]: def f(x):  
         return f(x-1)+1
```

```
In [17]: f(10)
```

```
-----  
--  
RecursionError: maximum recursion depth exceeded Traceback (most recent call last)  
Cell In[17], line 1  
----> 1 f(10)  
  
Cell In[15], line 2, in f(x)  
      1 def f(x):  
----> 2     return f(x-1)+1  
  
Cell In[15], line 2, in f(x)  
      1 def f(x):  
----> 2     return f(x-1)+1  
  
[...] [... skipping similar frames: f at line 2 (2975 times)]  
  
Cell In[15], line 2, in f(x)  
      1 def f(x):  
----> 2     return f(x-1)+1  
  
RecursionError: maximum recursion depth exceeded
```