List Comprehension

L = [1, 2, 3]

L2 = [item * 2 for item in L]

L3 = [item for item in L if item % 2 == 1]

condition (filter)

L4 = [True if item % 2 == 1 else False.

condition (if/else)

for item in L]

L = [. . . . .]

多少个 odd.

len([item for item in L if item % 2 == 1])

sum([item for item in L if item % 2 == 1] + [0])

→

any() any([□, ☆, ○, △])

↓

□ or ☆ or ○ or △

all([□, ☆, ○, △])

□ and ☆ and ○ and △

Recursion.

$$\sum_{n=1}^{100} i = 100 + \sum_{n=1}^{99} i$$

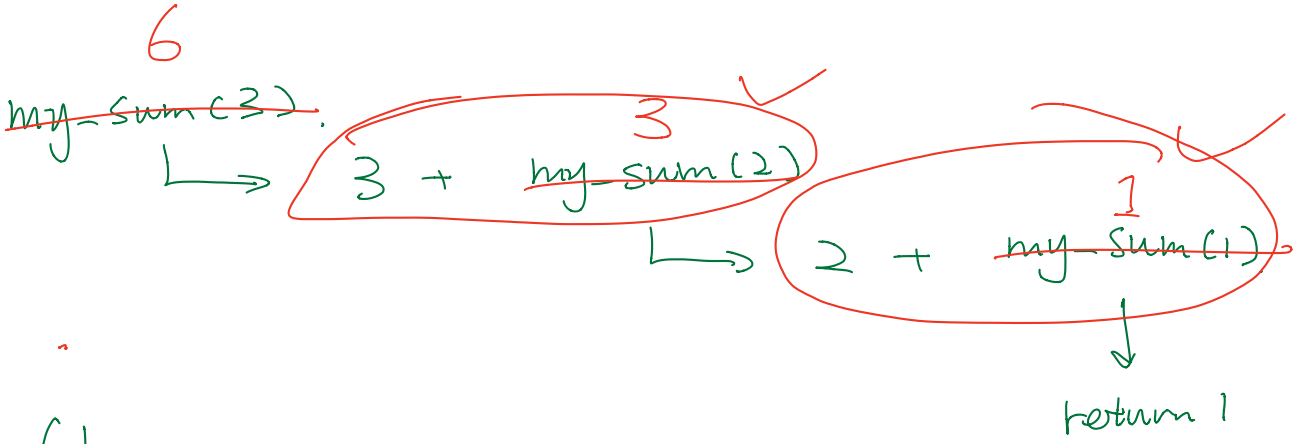
$$= 100 + 99 + \sum_{n=1}^{98} i$$

$= 100 + 99 + \cdots + \sum_{n=1}^{1} i$

$= 100 + 99 + \cdots + 1$

```
def my_sum(n):
    if n==1:        Base Case.
        return n

    else:
        return n + my_sum(n-1)
```



fib: 1 1 2 3 5 8

① function description:

② Base case.

③ divide.
combine.

```
def fib(n):
    if n<=2:
        return 1.

    else:
        return   fib(n-1) + fib(n-2).
```

Nested List Recursion. [□, ☆, △]

```
def my_sum(obj):
    if not isinstance(obj, list):
        return obj

    else.
        return sum([my_sum(item) for item in L])
```

[[5], 2]
7.

```
    acc = 0
    for sub in obj:
        acc += my_sum(sub)
    return acc
```

[[5], [3,2]]

3个 item.

```
def count(obj):
    if not isinstance(obj, list):
        return 1
```

[□, ⊗, ○]

```
    else:
        acc = 0
        for sub in obj:
            acc += count(sub)
        return acc
```

```
def depth(obj):
    if not isinstance(obj, list):
        return 0
    else:
        acc = []
        for sub in obj:
            acc.append(depth(sub))
        return max(acc + [0]) + 1
```

```
>>> depth(2)
0
>>> depth([2])
1
>>> depth([[2,3], 1])
2
```

[□, ⊗, ○]