2. 함수를 사용한 캣의 계산

```python
#[In]

def 계산(가격, 개수):
    return 가격 * 개수

print(계산(1000, 5), '원입니다.')
```

Python

가격과 개수를 매개변수로 입력하여 금액을 출력할 수 있습니다. 실행하면 아래와 같이 출력됩니다.

```python
#[Out]

5000 원입니다.
```

Python

> 등급 가격은 정해져 있는데 매번 입력하는 것은 귀찮다냥!

캣의 생선 가게에서는 등급에 따라 가격이 정해져 있기 때문에, 매번 값을 입력하는 것이 귀찮은 캣!
"매개변수와 연산을 수정하면 되겠다냥~"

```python
#[In]

# A등급:1000원, B등급:500원, C등급:100원

def 계산(a, b, c):
    합계 = a*1000 + b*500 + c*100
    return 합계

print(계산(5, 2, 3), '원입니다.')
```

Python

위 코드를 실행하면 아래와 같이 출력됩니다.

```python
#[Out]

6300 원입니다.
```

Python

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

2.1 상수(Constant)

상수(Constant)란 **변경할 수 없는 수**입니다. C언어에서는 define이라는 문장을 사용하여 상수를 정의하지만, Python에서는 이러한 상수를 선언하는 문법이 없습니다. 다만 **일반적으로 이름 전부를 대문자로 선언**하면 상수라고 이해합니다.

```python
#[In]

PI = 3.14
def circle(r,inputpi):
    z = r*r*inputpi
    return z
result = circle(10, PI)
print(result)
```

Python

위 코드를 실행시키면 아래와 같이 출력이 됩니다.

```python
#[Out]

result : 314.0
```

Python

여기서는 `PI = 3.14` 이 상수가 됩니다. 한번 더 주의 깊게 보아야 할 내용은 함수에서 인자값으로 받는 변수가 `PI` 가 아니라 `inputpi` 라는 것입니다. 이 개념에 대해서는 다음장 지역변수와 전역변수를 다루며 말씀드리도록 하겠습니다.

```python
#[In]

π = 3.14
def circle(r, inputpi):
    z = r*r*inputpi
    return z
result = circle(10,π)
print(result)
```

Python

이처럼 특수문자로도 사용이 가능합니다. 같은 원리로 한글 코딩도 가능합니다. 한글 코딩의 장점으로는 고유명사를 표현하기 좋으며, 교육용으로도 주목받고 있습니다.

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

HTML, CSS, Javascript등 다양한 분야에서 한글코딩이 주목받고 있으며 자세한 내용은 '한글코딩.org'에서 확인할 수 있습니다. 아래는 한글코딩 예시입니다.

```python
#[In]

def 한글코딩(글자):
    print(글자)
한글코딩('안녕, 세상아.')
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```python
#[Out]

안녕, 세상아
```

Python

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

3. 전역변수 global

지역변수는 함수 내부에서만 사용되는 변수입니다. 함수가 호출되는 동안에만 효력을 발휘하고 함수가 끝나는 동시에 소멸됩니다.

전역변수는 프로그램 어디에서나 접근이 가능한 변수이며 함수 내에서 **global**을 입력하면 전역변수가 됩니다.

```python
#[In]

A등급 = 0

def 낚시():
    global A등급
    A등급 += 1

낚시()
낚시()
낚시()

print('잡은 A등급 :', A등급, '마리')
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```python
#[Out]

잡은 A등급 : 3 마리
```

Python

위 코드에서는 함수가 호출될 때마다 전역변수 A등급에 1씩 더합니다. 총 3번 호출되었으므로 전역변수 A등급에는 3이 저장됩니다.

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

이번에는 낚시, 그물, 통발을 모두 사용하여 잡은 생선의 총 합계를 구해봅시다.

```python
#[In]

# 캣의 생선 잡기

A = 0
B = 0
C = 0
문어 = 0

def 낚시():
    global A
    A += 1

def 그물():
    global A, B, C
    A += 3
    B += 3
    C += 4

def 통발():
    global 문어
    문어 += 1

낚시()
그물()
통발()

합계 = A + B + C + 문어
print('오늘 잡은 생선의 합계 :', 합계)
```

Python

위 코드에서는 모든 함수에서 전역변수를 사용하기 때문에 함수가 변수의 값에 영향을 줍니다. 실행시키면 아래와 같이 출력됩니다.

```python
#[Out]

오늘 잡은 생선의 합계 : 12
```

Python

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

지역변수와 전역변수를 왜 나누어 놓았나요?

예를 들어 물고기를 어떻게 잡았는지 구분하는 업무를 하게 되었다고 가정해 보겠습니다. 캣은 통발로 잡았을 때 x라는 변수를 사용하고 캣의 직원은 낚시로 잡았을 때 x라는 변수를 사용하는데, 이 변수를 자신이 만든 프로그램이 아닌 다른 프로그램에서 조작이 되어진다면 프로그램을 만드는 과정에서 혼선이 생길 수 있습니다.

따라서 이러한 혼선을 막기 위해서 프로그래밍 언어에서는 **함수 내에서 선언된 변수와 구현 내용을 외부에서 만지지 못하도록 지원**하고 있습니다.

```python
#[In]

# global 사용전
a = 100
def f():
    a = a + 1
f()
```

Python

```python
#[In]

# global 사용후
a = 100
def f():
    global a
    a = a + 1
f()
```

Python

함수 안에서는 함수 밖에 있는 변수에 접근하지 못합니다. 이를 위해 **global**이라는 함수를 사용하여 모든 함수에서 사용 가능하도록 만들 수 있어요.

> 💡 그러나 프로그래밍에서 전역변수는 권장하지 않습니다. 연산을 하고 싶다면 함수에 아규먼트(함수의 input)를 대입하여 return으로 받는게 객체 지향 프로그래밍에 적합합니다.

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

4. function 응용

4.1 함수 안에 함수 만들기

```
def 함수이름1():
    코드
    def 함수이름2():
        코드
```

Python

이번에는 위 코드처럼 **함수 속에 함수**를 만드는 방법입니다. 위와 같이 def 로 함수를 만들고 그 안에 다시 def로 함수를 만들면 됩니다.

```
#[In]

def print_text():
    text = '생선을 잡아보자!'
    def txt():
        print(text)
    txt()

print_text()
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

생선을 잡아보자!
```

Python

함수 `print_text()` 안에서 다시 def로 함수 `txt()`를 만들었습니다. 그리고 `print_text()`안에서 `txt( )`처럼 함수를 호출했습니다. 하지만 아직 함수를 정의만 한 상태이므로 아무것도 출력되지 않습니다.

두 함수가 실제로 동작하려면 바깥쪽에있는 `print_text()`를 호출해주어야 합니다. 즉, `print_text()` → `print()` 순으로 실행됩니다.

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

그러면 함수 안에 여러개의 함수를 만들 수도 있을까요? 네, 가능합니다.

```python
#[In]

def 생선잡기():
    물고기 = {'A등급':0, 'B등급':0, 'C등급':0}

    def 낚시():
        물고기['A등급'] += 1

    def 그물():
        물고기['A등급'] += 3
        물고기['B등급'] += 3
        물고기['C등급'] += 4

    def 통발():
        물고기['문어'] = 1

    낚시()
    그물()
    통발()

    return 물고기

생선잡기()
# sum(생선잡기().values())
```

Python

위 코드를 실행하면 아래와 같이 출력됩니다.

```python
#[Out]

{'A등급': 4, 'B등급': 3, 'C등급': 4, '문어': 1}
```

Python

함수 `생선잡기()` 안에서 함수 `낚시()` 와 함수 `그물()` , `통발()`을 만들었습니다. 그리고 `생선잡기()` 는 세 함수를 호출하여 각 함수의 기능들을 수행하게 됩니다.

이 예제에서도 마찬가지로 함수들이 실제로 동작하기 위해서는 함수 밖에서 `생선잡기()` 를 호출해주어야 합니다.

내가 나를 호출하는 **재귀함수**에 대해서는 깊은 물에서 알아보도록 하겠습니다! 아래 간단한 재귀함수에 대한 코드가 있는데요. 이렇게 사용할 수 있구나 정도를 파악하고 가시면 좋을 것 같아요.

```python
def factorial(x):
    if x == 1:
        return 1
    else:
        return x * factorial(x-1)

factorial(5)
```

Python

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO

5. 연산자 우선순위

연산자 우선순위는 앞서 <**2편 생선을 잡아라**>에서 알아보았습니다. 아래 표에 추가된 내용을 한 번 살펴보고 넘어가시길 바랍니다.

다음은 파이썬의 연산자 우선순위입니다.

연산자 우선순위

우선순위	연산자	설명
1	(값...), [값...], {키:값...}, {값...}	튜플, 리스트, 딕셔너리, 세트 생성
2	x[인덱스], x[인덱스:인덱스], x(인수...), x 속성	리스트(튜플) 첨자, 슬라이싱, 함수 호출, 속성 참조
3	await x	await 표현식
4	**	거듭제곱
5	+x, -x, ~x	단항 덧셈(양의 부호), 단항 뺄셈(음의 부호), 비트 NOT
6	*, @, /, //, %	곱셈, 행렬 곱셈, 나눗셈, 버림 나눗셈, 나머지
7	+, -	덧셈, 뺄셈
8	<<, >>	비트 시프트
9	&	비트 AND
10	^	비트 OR
11	|	비트 XOR
12	in, not in, is, is not <, <=, >, >=, ==, !=	포함 연산자, 객체 비교 연산자, 비교 연산자
13	not x	논리 NOT
14	and	논리 AND
15	or	논리 OR
16	if else	조건부 표현식
17	lamda	람다 표현식

+ New

COUNT 17

Project name :

DATE .

비고	Line	File name :				
	1					
	2					
	3					
	4					
	5					
	6					
	7					
	8					
	9					
	10					
	11					
	12					
	13					
	14					
	15					
	16					
	17					
	18					
	19					
	20					
	21					
	22					
	23					
	24					
	25					
	26					
	27					
	28					
	29					
	30					

MEMO