7. 멤버 연산자

멤버 연산자는 **특정 항목이 배열에 들어있는지 확인하는 연산**입니다. 결과값이 참이면 **True**, 거짓이면 **False**를 반환합니다.

배열 `오늘_잡은_물고기` 에는 'A등급'이 있기 때문에 in 연산의 결과값은 True, not in 연산의 결과값은 False가 출력됩니다.

```python
#[In]

# 오늘 A등급 물고기를 잡았나요?
오늘_잡은_물고기 = ['A등급', 'A등급', 'B등급', 'C등급', 'C등급', 'C등급']

print('A등급' in 오늘_잡은_물고기) # True
print('A등급' not in 오늘_잡은_물고기) # False
```

Python

배열 `오늘_잡은_물고기` 에는 'D등급'이 없기 때문에 in 연산의 결과값은 False, not in 연산의 결과값은 True가 출력됩니다.

```python
# 오늘 D등급 물고기를 잡았나요?
오늘_잡은_물고기 = ['A등급', 'A등급', 'B등급', 'C등급', 'C등급', 'C등급']

print('D등급' in 오늘_잡은_물고기) # False
print('D등급' not in 오늘_잡은_물고기) # True
```

Python

멤버 연산자

✨ in	왼쪽 항목이 오른쪽 배열에 포함되면 True 그렇지 않으면 False
✨ not in	왼쪽 항목이 오른쪽 배열에 포함 안 되면 True 그렇지 않으면 False

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

8. 식별 연산자

식별 연산자는 **해당 값이 들어있는** `주소` **를 비교하는 연산자**입니다. C, C++ 언어를 학습하신 분이라면 포인터로 이해하시면 됩니다.

그렇지 않으신 분을 위해 간단히 설명하자면, **변수를 선언해서 값을 저장할 때 그 값을 저장하는 공간이 필요한데 그 공간의 위치를 주소**라고 합니다. 저희들이 사용하는 주소와 비슷한 개념이라고 보시면 됩니다. OO시 OO동 같은 주소 체계를 컴퓨터에서도 가지고 있는 것입니다.

식별 연산자

✨ is	피연산자들의 위치(주소)가 같다면 True 그렇지 않으면 False
✨ is not	피연산자들의 위치(주소)가 다르면 True 그렇지 않으면 False

`위니브윌드` 에서 `licat` 이라는 사람이 일하고 있다고 가정합시다. 또 `바울랩` 이라는 회사에도 `licat` 이라는 사람이 일하고 있을 경우 이 둘이 동일인물인지 확인해봅시다.

```python
#[In]

위니브윌드 = "licat"
바울랩 = "licat"

print(위니브윌드 is 바울랩)  # True
print(위니브윌드 == 바울랩)  # True
```

Python

같다면 동일인물이라는 것을 알 수 있습니다.

즉, 이 두 회사에 다니고 있는 `licat` 이라는 이름을 가진 사람은 `OO시 OO동 XX번지` 에서 살고 있는 동일인물이라고 생각하면 됩니다.

> 💡 물론 실제로 컴퓨터 내에서의 주소는 이런 주소가 아닌 0012FF64 와 같은 16진수 주소 체계를 가집니다.

이때 그럼 `==` 와 차이가 뭔지 궁금해할 수 있습니다. `==` 는 값이 같은지 찾는 비교 연산자입니다. 위의 예시로 보자면 같은 `licat` 이라는 이름을 가졌는지를 보는 것입니다.

다음 예시를 보면서 확인해 보겠습니다.

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

```python
#[In]

a = [1, 2, 3]
b = [1, 2, 3]
c = a

print('a == b', a == b)
print('a == c', a == c)
print('a is b', a is b)
print('a is c', a is c)
```

Python

```python
#[Out]

a == b True
a == c True
a is b False
a is c True
```

Python

위에서는 `licat` 이라는 문자열을 예시로 들었습니다. 파이썬의 경우 동일한 문자열이 있다면 해당 문자열을 가진 주소를 그대로 참조하게 되어있습니다. 그렇기 때문에 동일한 문자열을 생성하는 경우에는 동일한 주소를 보고 있게 됩니다. (항상 그런 것은 아닙니다.)

배열과 같은 객체는 문자열과 달리 새로 생성할 때 새로운 주소에 담습니다. 따라서 값은 동일하더라도 다른 주소를 가지게 될 수 있습니다.

따라서 위의 예시와 같이 `a`, `b`, `c`의 값을 비교하는 것은 `True`로 나오지만 주소를 비교했을 때는 다르게 나오는 것을 볼 수 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

9. 연산자의 우선순위

연산자의 우선순위는 어떤 연산을 먼저 할 것인지 정하는 규칙입니다. 따라서 우선순위가 높은 연산자부터 연산을 진행합니다.

연산자의 우선순위

Aa 순위	연산자 기호	설명
1	(), {}, []	Tuple, Set, List, Dictionary
2	**	거듭제곱
3	+, -, ~	단항 연산자
4	*, /, //, %	곱하기, 나누기, 몫(정수), 나머지
5	+, -	더하기, 빼기
6	<<, >>	쉬프트 연산
7	&	비트연산 AND
8	^, |	비트연산 XOR, 비트연산 OR
9	in, not in, is, is not, <, <=, >, >=, ==, !=	멤버 연산자, 식별 연산자, 비교 연산자
10	not	논리 부정
11	and	논리 AND
12	or	논리 OR

COUNT 12

연산자의 우선순위는 중요한 개념이나 위 표를 보고 모두 익히기에는 어렵습니다.

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

먼저, 사칙연산의 우선순위는 대부분 다 아실 것입니다. (`+` , `-` , `*` , `/`) 곱셈과 나눗셈이 우선이고, 덧셈과 뺄셈이 그 다음 순위입니다.

그 외의 연산자들은 천천히 사용하면서 익혀보며 혼란이 있을 경우에는 가장 높은 우선순위인 괄호 `()` 를 이용하여 묶어서 처리하는 것이 더 직관적일 수 있습니다.

```python
#[In]

a = 10 + 3 * 5
print(a)  # 25

a = (10 + 3) * 5
print(a)  # 65
```

Python

이렇게 괄호를 사용하면 우선순위가 달라져 연산의 결과가 달라질 수 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

캣의 해골섬 출항!

PYTHON BOOTCAMP

1. **캣의 통장에 노드(위니브 월드의 통화단위)를 저축하시오!**

   캣은 물고기를 잡아 파는 행위를 반복하였어요. 어느새 캣의 통장에는 100만 노드라는 돈이 쌓였습니다. 또, 매일 스킬이 오르고, 요령도 생겨 물고기를 10마리씩 더 잡게 되었습니다.

   물고기의 가격은 30노드이며 이번달 1일에는 110마리, 2일에는 120마리, 3일에는 130마리 씩 10일간 물고기를 잡아 팔았다고 했을 때 캣의 통장에는 얼마의 돈이 쌓여 있을까요?

```python
캣의_통장 = 1000000
물고기의_가격 = 30
잡은_물고기의_수 = '정답을 입력하세요.'
물고기를_팔고_난_후_캣의_통장 = '정답을 입력하세요.'
```

2. **물고기가 잘 잡히는 반경(넓이)을 구하시오!**

   캣은 물고기를 잡으러 갈 때마다 물고기가 잘 잡히는 곳이 있다는 사실을 알게 되었습니다. 물고기가 잘 잡히는 곳은 저 멀리 해골섬 주위로 반경 500m의 큰 원을 그리고 있습니다.

```python
반지름 = 500
고기가_잘_잡히는_반경 = '정답을 입력하세요.'
```

3. **물고기를 다 잡고난 후 캣의 통장 잔고를 구하시오!**

   원의 1 넓이당 물고기가 110마리씩 살고 있습니다. 해골섬의 넓이는 314입니다. 이 물고기를 다 잡았을 때 통장 잔고에 얼마가 있을지 구하세요. 현재 통장 잔고는 '물고기를_팔고_난_후_캣의_통장'입니다.

```python
해골섬의_넓이 = 314
반지름 = 500
물고기를_다_잡고_난_후_캣의_통장 = '정답을 입력하세요.'
```

4. **번 돈과 비율을 정리하시오!**

   A등급 물고기와 B등급 물고기가 각각 100노드, 50노드이고 운이 좋게도 A등급은 350마리, B등급은 700마리를 잡았다면 총 얼마의 돈을 벌었을까요? 그리고 총액에서 A등급이 차지하는 비율과 B등급이 차지하는 비율을 구하십시오. 여기서 비율은 Dictionary로 구하십시오.

```python
물고기_등급 = {'A등급':100, 'B등급':50}
총액 = '정답을 입력하세요.'
비율 = '정답을 입력하세요.'
```

모두 완료하였다면 아래 훈장을 추가하세요!

```python
훈장.append('해골섬 낚시꾼')
```

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

3편 효율적으로 생선 잡기

1. 캣의 고민

해골섬에서 잡은 물고기는 살이 통통하고 맛이 일품이라 인기가 날이 갈수록 높아졌습니다. 심지어 웃돈을 주고 생선을 사기까지 이르렀어요. 캣은 넘쳐나는 수요에 깊은 고민에 빠졌습니다.

"더 효율적으로 많은 생선을 잡을 방법이 없을까냥?"

이때 지나가던 상인이 작게 귀띔을 해주었습니다.

"위니브 왕국에서는 아주 큰 그물과 통발을 사용하여 생선을 많이 잡아들인다는 소문이 있소냥~"

캣은 조언에 귀를 기울이고 큰 그물과 통발을 구하러 다녔습니다. 그리고 그 사용법을 익히기 시작했어요.

"여러 도구를 사용하여 다양한 방법으로 생선을 잡아봐야겠다냥!"

```python
#[In]

# 현재 캣이 보유한 기술
# 기술 = ['고기잡이', '고기팔기']
기술.append('낚시_Lv1')
기술.append('통발_Lv1')
기술.append('큰그물_Lv1')
```

Python

- **라이캣의 현재 상태창!**

```python
#[In]

이름 = '캣'
설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
나이 = 15
오늘_잡은_물고기 = '1000'
키 = '45.5cm'
몸무게 = 1.3
육식 = True
초식 = True
돈 = 1000
훈장 = ['백상아리를 잡은 고양이', '성실한 납세자', '해골섬 낚시꾼']
기술 = ['고기잡이', '고기팔기', '낚시_Lv1', '통발_Lv1', '큰그물_Lv1']
```

Python

- 규칙 -
1. 낚시는 A등급 생선을 한 마리씩 잡을 수 있습니다.
2. 그물로는 한 번에 A등급 3마리, B등급 3마리, C등급 4마리를 잡을 수 있습니다.
3. 통발은 하루에 한 번만 확인하며 문어 한 마리를 잡을 수 있습니다.

```python
#[In]

# 캣의 생선 잡기
# 함수 정의하기
def 낚시():
    print('A등급 생선을 한 마리 잡았습니다.')

def 그물():
    print('A등급 3마리, B등급 3마리, C등급 4마리를 잡았습니다.')

def 통발():
    print('문어를 하나 잡았습니다.')

# 함수 호출하기
낚시()
그물()
통발()
```

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

여기서 `Alt + Enter` 를 누르면 아래와 같이 출력됩니다.

```
#[Out]

A등급 생선을 한 마리 잡았습니다.
A등급 3마리, B등급 3마리, C등급 4마리를 잡았습니다.
문어를 하나 잡았습니다.
```

Python

1.1 들여쓰기

Python에서는 **들여쓰기(intend)로 함수의 범위**를 정합니다. 함수의 범위를 규정하는 것은 화이트 스페이스(공백)입니다. 탭으로 한번에 들여 쓸 수 있지만 파이썬 개발 제안서(PEP 8)에서는 **스페이스 4번**으로 하기로 약속했으니 스페이스를 사용해주시길 바랍니다.

```
#[In]

def 낚시():
    print('A등급 생선을 한 마리 잡았습니다.')
    print('B등급 생선을 한 마리 잡았습니다.')

낚시()
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

A등급 생선을 한 마리 잡았습니다.
B등급 생선을 한 마리 잡았습니다.
```

Python

```
#[In]

def 낚시():
    print('A등급 생선을 한 마리 잡았습니다.')

print('B등급 생선을 한 마리 잡았습니다.')
낚시()
```

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

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

B등급 생선을 한 마리 잡았습니다.
A등급 생선을 한 마리 잡았습니다.
```

Python

💡 이처럼 들여쓰기를 통해 함수의 범위를 다르게 하면 결과가 달라지므로 함수를 사용할 때는 들여쓰기에 유의해야 합니다.

1.2 함수의 사용

```
#[In]

def function(x, y):
    z = x + y
    return z

# A등급 10마리, B등급 9마리
print('오늘 잡은 생선 :', function(10, 9))
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

오늘 잡은 생선 : 19
```

Python

이처럼 **입력과 기능 및 연산, 출력 값이 있는 것을 '함수'**라고 말합니다. 함수를 선언하였다면 호출해야만 함수가 작동하게 됩니다.

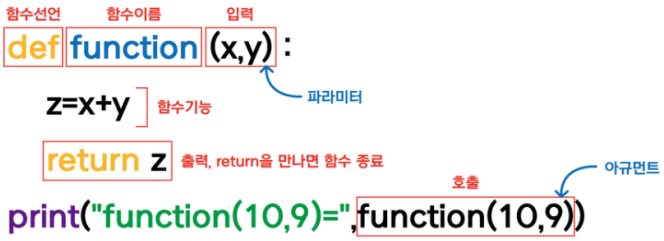
Project name :

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

위 예제에서는 function(10, 9)라는 문장으로 함수를 호출합니다.



입력과 출력값 등 함수의 모든 요소가 필요충분조건인 것은 아닙니다.

```python
#[In]

def 낚시():
    print('A등급 생선을 한 마리 잡았습니다.')
```

Python

위 예제의 함수 `낚시()` 와 같이 input이 없는 함수도 있고, return 값이 없는 함수도 있습니다. return 값이 없는 함수는 자동으로 None이 할당됩니다. 또한 function이 없는 함수도 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

1.3 print VS return

```python
#[In]

def function(x, y):
    z = x + y
    print(z)

print('오늘 잡은 생선 :', function(10, 9))
```

Python

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

19
오늘 잡은 생선 : None
```

Python

위 예제에서 `function(10, 9)`를 호출한 값을 출력해 보았더니 None이 출력됩니다. 이는 이 함수의 return 값이 없기 때문입니다.

💡 종종 print와 return을 헷갈려 하시는 분들이 계십니다. print는 다른 함수라는 걸 기억해주세요!

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