

PYTHON BOOTCAMP

· 얹은물 ·





PYTHON BOOTCAMP

파이썬 기초 강의인 <파이썬 부트캠프>는 비전공자를 위한 '얕은 물' 강의와 전공자를 위한 '깊은 물' 강의로 이뤄져 있습니다. 두 강의는 같은 문법을 설명하더라도 난이도에 따라 설명하는 방식이 다릅니다.

'얕은 물' 강의는 스토리를 활용하여 쉽고 재미있게 파이썬을 학습할 수 있으며, '깊은 물' 강의는 스토리 없이 담백하게 핵심을 파고들 수 있도록 구성되어 있습니다.

이 책은 '파이썬 부트캠프 - 얕은물' 강의 내용과 라이캣의 성장 스토리를 담고 있습니다. Python으로 라이캣을 성장시키고, 그의 행보에 동력자가 되십시오. 연습문제를 통해 여러 난관들을 극복해 가세요.

여러분의 성장을 응원합니다.

이 책은 인쇄용으로 사용하시고, 아래 링크에서 노션(Notion)으로 된 Code Page를 보실 수 있습니다. 실습을 하실 때에는 노션에서 Code를 복사해 사용하세요.



[PYTHON BOOTCAMP - 얕은물] 노션 페이지 링크

<http://bitly.kr/FZeIYN0qa0W>



이 책에 있는 'PYTHON BOOTCAMP - 얕은물' 강좌는 인프런에서 만나보실 수 있습니다.

저자 소개

이호준

現 바울랩아이씨티기술연구원 대표
現 바울랩아이씨티컴퓨터학원 대표
現 사도출판 대표
現 바울랩미디어 대표
現 주식회사 위니브 대표
現 GDG(Google Developers Group) JEJU Organizer
現 제주스타트업협회 부회장
現 제주대학교 폴스택 수업 강사
現 제주코딩베이스캠프 운영진

“ 직책과 감투가 많지만, 사회에 공헌하며 미래를 꿈꾸는 아이들 가르치며 소박하게 살고 싶은 제주 도민입니다.”

김혜원

前 바울랩아이씨티기술연구원 대표
現 주식회사 위니브 CTO · COO

카카오와 함께하는 제주 코딩 베이스캠프 11기 스태프
코딩도장 튜토리얼로 배우는 Python 문제풀이 외 4권 집필

김유진

前 바울랩아이씨티기술연구원 대표
現 주식회사 위니브 CTO · COO

제주코딩베이스캠프 Code Festival : Python 100제 집필

차경림

前 제주대학교 산업디자인학부 문화조형디자인전공
現 주식회사 위니브 CDO

저자 소개

김난화

現 제주대학교 컴퓨터공학전공
現 바울랩아이씨티기술연구원 연구원

김승민

現 제주대학교 사범대학 컴퓨터교육과
現 바울랩아이씨티기술연구원 연구원

김 진

現 제주대학교 사범대학 컴퓨터교육과
現 바울랩아이씨티기술연구원 연구원

이승신

前 제주대학교 언론홍보학과
現 제주 더큰내일센터 '탐나는 인재' 1기
現 바울랩아이씨티기술연구원 연구원

이진우

現 제주대학교 사범대학 컴퓨터교육과
現 바울랩아이씨티기술연구원 연구원

제주코딩베이스캠프 로드맵





jeju coding basecamp

수료증 제도



제코베 수료증 제도란?

제주코딩베이스캠프 강좌를 **20학점 이상, 90% 이상** 수료시
수료증 및 각종 혜택을 제공해 드립니다.

학점 안내

제주코딩베이스캠프 인프런 강의



1학점



2학점



3학점



5학점



1학점

기타 강의 및 행사

제코베 강좌를 제외한 IT강의(플랫폼 제한 없음)

1학점/최대 5학점

제코베 오프라인 강의 및 행사 참여

5학점



혜택
1
인재풀 등록 및
취업 지원



혜택
2
수료증 및
제코베 굿즈 키트



혜택
3
강의 공동 제작
책 공동 집필 기회

참여 방법

1. 제코베 온라인 강좌, 기타강의 및 행사를 20학점 이상, 90% 이상 수강/참여한다.
2. 증빙 자료(캡처)를 구글 설문지를 통해 제출한다.
3. 취업 지원 또는 포트폴리오 인증서 발급을 원할 경우 포트폴리오/Github/블로그 URL 등을 구글 설문지를 통해 제출한다.

공지 사항

- 제코베 기본 강좌인 '코알못에서 웹서비스 런칭까지 : 제주코딩베이스캠프'는 필수로 수강해야 합니다.
- 포트폴리오 인증서는 수료증을 발급 받게 되는 분들을 대상으로 하며, 원할 경우에 포트폴리오 검수 후 발급해 드립니다.



Contents

얕은물 강좌 스토리 미리보기 11

환경설정 14

- 1. Google Colab 간단한 사용법!
- 2. 상세한 사용법!

1편 라이캣의 탄생! 19

- 1. 캣의 일상
- 2. 캣의 정보 변경
- 3. 각 변수들의 타입을 알아보기
- 4. 각 변수들의 속성을 알아보기
 - 4.1 int의 속성 알아보기
 - 4.2 float의 속성 알아보기
 - 4.3 str의 속성 알아보기
 - 4.4 bool의 속성 알아보기
 - 4.5 list, tuple의 속성 알아보기
 - 4.6 dict의 속성 알아보기
 - 4.7 set의 속성 알아보기
 - 4.8 list, tuple, set, dict에 대해 더 알아 보기
- 5. 각 변수들을 형변환 해보기
- 6. 출력을 하는 여러가지 용법

2편 생선을 잡아라 64

- 1. 캣의 다짐
- 2. 캣의 생선팔기
- 3. 대입 연산자(할당 연산자)
- 4. 비교 연산자
- 5. 논리 연산자
- 6. 비트 연산자
- 7. 멤버 연산자
- 8. 식별 연산자
- 9. 연산자의 우선순위

캣의 해골섬 출항! 106

3편 효율적으로 생선 잡기 110

- 1. 캣의 고민
 - 1.1 들여쓰기
 - 1.2 함수의 사용
 - 1.3 print VS return
- 2. 함수를 사용한 캣의 계산
 - 2.1 상수(Constant)
- 3. 전역변수 global
- 4. function 응용
 - 4.1 함수 안에 함수 만들기
- 5. 연산자 우선순위



4편 생선 회사를 운영하라 141

- 1. 캣의 생선회사
 - 1.1 캣의 회사 운영
 - 1.2 if 문의 기본 구조
- 2. if, else 문
- 3. if, elif, else

163 **캣의 고객관리**

171 **5편 라이캣의 EXIT**

- 1. 캣의 로봇(for)
 - 1.1 for문의 기본 구조
 - 1.2 for, else
 - 1.3 지능형 리스트(list comprehension)의 for
 - 1.4 다중인자 리스트 순회
 - 1.5 enumerate
 - 1.6 계산하는 로봇
- 2. 캣의 로봇(while)
 - 2.1 무한반복 while, break
 - 2.2 while, else
- 3. break
 - 3.1 break 문
- 4. continue, pass
- 5. else
- 6. 중첩 반복문

246 **6편 파이와 썬의 알고리즘 7원석을 찾아서**

- 1. 라이캣의 모험
- 2. 클래스
 - 2.1 클래스 변수와 인스턴스 변수
 - 2.2 _init_ 함수
 - 2.3 상속
 - 2.4 다중 상속
 - 2.5 특별 메소드(magic method)
 - 2.6 캣의 준비물



240 **사자탈을 쓴 캣**

270 **캣의 모험 자금을 모아라!**

244 **스토리 : 쏘아진 화살!**

277 **파이와 썬이 남긴 단 하나의 단서**





얇은물 강좌 스토리 미리보기

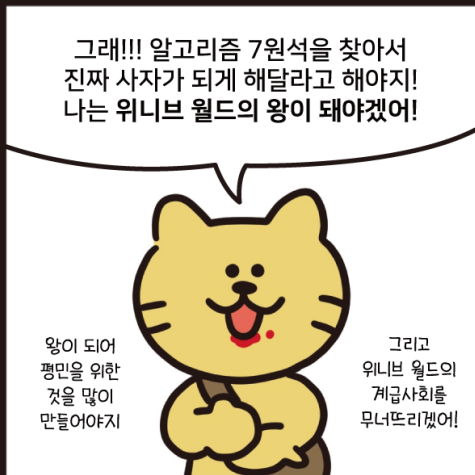
Python 부트캠프

얕은물 강좌 스토리 미리보기





▶ 반복 : 라이캣의 EXIT



▶ 자료형 정리 : 파이와 썬의 알고리즘 7원석을 찾아서

스토리는 제코베 강의 '눈떠보니 코딩 테스트 전날'과 이어집니다.



환경설정

1. Google Colab 간단한 사용법!
2. 상세한 사용법!



Google 아이디만 있다면 사용할 수 있는 Google Colab!

여러가지 불편하게 설치하는 작업 없이 여러분의 빠른 위니브 월드 진입을 도와줄 것입니다.

1. Google Colab 간단한 사용법!

1. google에 로그인을 해주시고, google에서 google colab이라고 검색을 합니다.

Google

google colab

전체 뉴스 동영상 이미지 도서 더보기

검색결과 약 3,330,000개 (0.35초)

colab.research.google.com ▾ 이 페이지 번역하기

Google Colab

Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your ...

[Introduction to Colab and Python](#) · [Tensorflow with GPU](#) · [Overview of Colaboratory](#)

이 페이지를 여러 번 방문했습니다. 최근 방문 날짜: 20. 6. 16

colab.sandbox.google.com ▸ notebooks ▾

Colaboratory란? - Google Colab

Google 드라이브 계정에서 스프레드시트를 비롯한 데이터를 Colab 메모장으로 가져오거나 GitHub 등의 여러 다른 소스에서 데이터를 가져올 수 있습니다. Colab을 ...

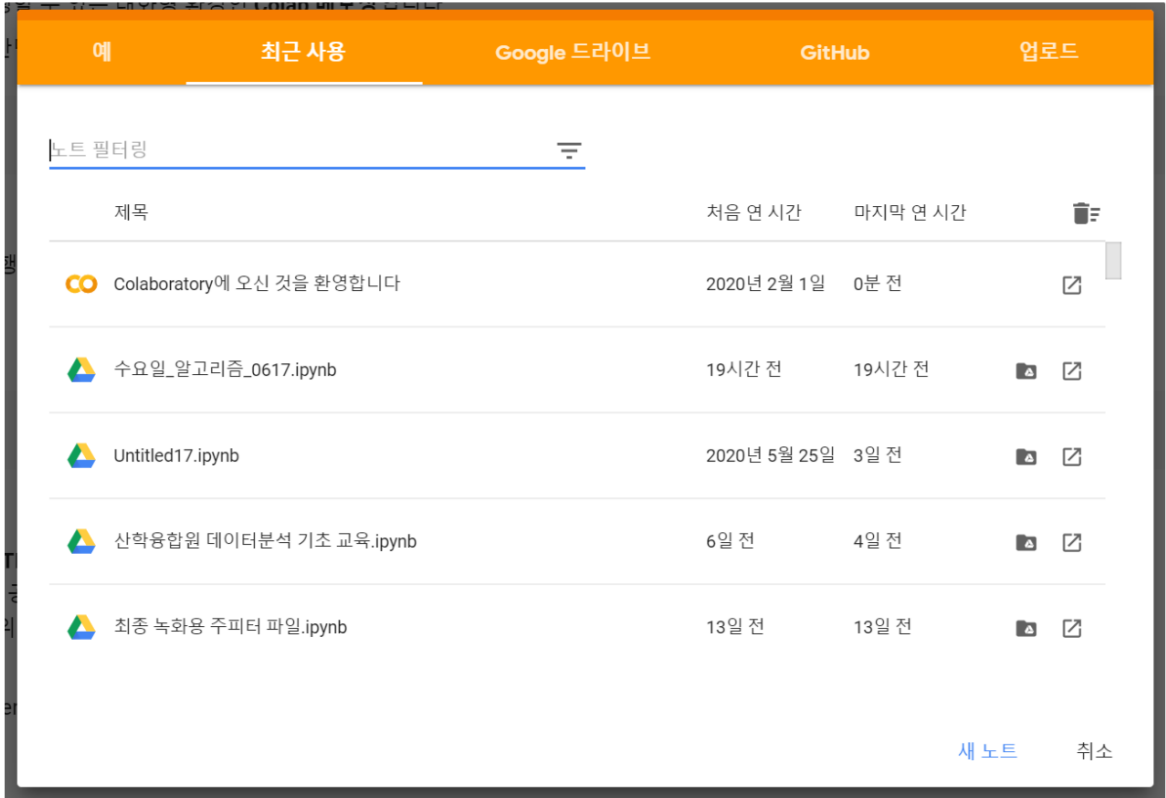
zszsa.github.io ▸ data ▸ 2018/08/30 ▸ google-colab ▾

Google Colab 사용하기 · 어쩐지 오늘은

2018. 8. 30. - Google의 Colab 사용법에 대해 정리한 글입니다 이 글은 계속 업데이트 될 예정입니다! 목차 UI 상단 설정 구글 드라이브와 Colab 연동 구글 ...

구글 드라이브와 Colab 연동 · 구글 드라이브와 로컬 연동 · Github 코드를 Colab에서 ...

2. 다음과 같이 문서를 설정하는 창이 뜨는데요. 여기서 **새노트를 클릭**해주세요. 기존에 사용하신 파일이 있다면 업로드 해서 편집도 가능합니다. 그리고 **모든 파일은 자동으로 google drive에 저장**됩니다!



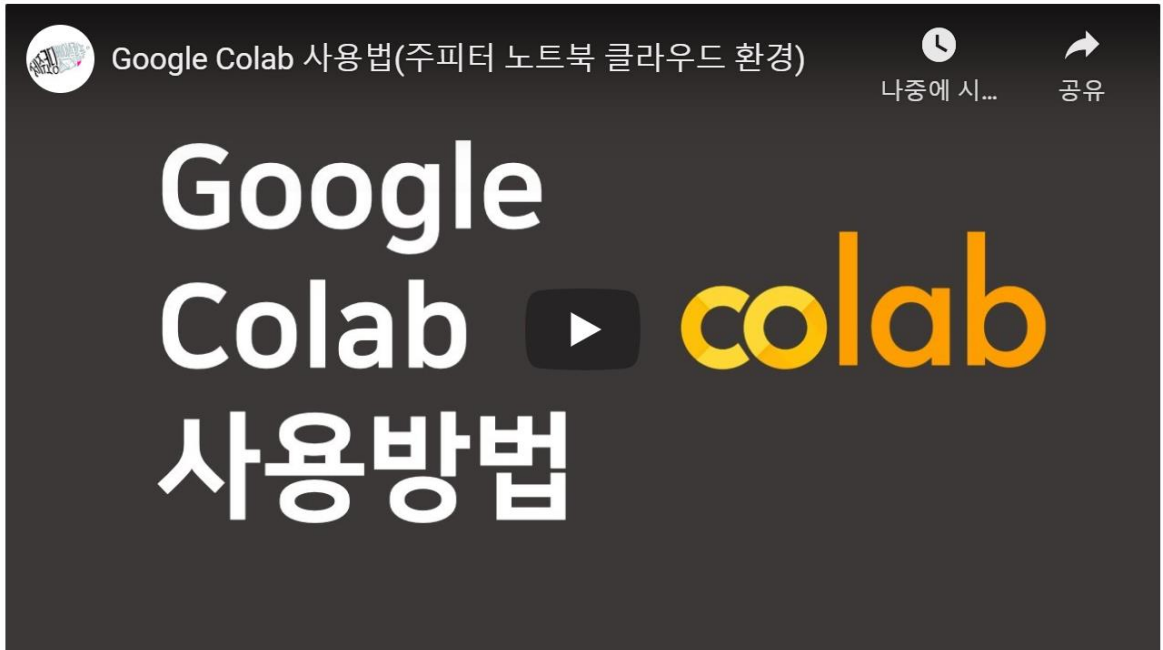
3. 아래와 같이 새 노트가 만들어집니다. 메뉴를 설명해 드리도록 하겠습니다!

- 맨 위 제목을 설정하는 창이 있습니다. 누르시면 파일의 제목을 수정할 수 있어요. 오른쪽에 보시면 누군가와 파일을 공유할 수도 있답니다.
- 파일, 수정, 보기, 삽입, 런타임, 도구, 도움말 등의 상태창이 떠요. 이걸 지금 단계에서 필요하진 않으니 오른쪽에 위로 화살표가 그려진 것을 눌러 접어주세요. (상세 기능 설명은 아래 동영상 참고하세요. 그러나, 초반에 너무 상세한 내용은 오히려 즐거운 코딩을 방해합니다.)
- 그 아래 재생 버튼을 누르면 코드를 실행할 수 있어요! 단축키로는 **Ctrl + Enter** 입니다. 이 단축키는 많이 사용하니 알고 있으셔야 합니다!
- 그 아래 코드를 삽입할 수도 있어요. 일반 텍스트를 삽입하여 설명을 추가할 수도 있죠. 일반 텍스트는 **마크다운** 을 활용하고 있어요. 코드를 실행시키고 바로 코드 셀을 추가하는 명령어는 **Alt + Enter** 입니다. 역시 많이 쓰는 단축키니 꼭 기억해두세요.



2. 상세한 사용법!

1. colab : 처음에 말씀드렸듯이, 초반에 접하는 상세한 내용은 즐거운 코딩을 방해합니다. 가능하다면 이 챕터를 건너뛰시고, 어느정도 적응이 된 후 다시오세요. 😊



2. jupyter notebook : 로컬에서 jupyter notebook을 설치하시면 조금 더 쾌적한 환경에서 개발이 가능합니다. 그렇지만, 기억하세요. 구글 colab은 무려 Ram 할당량이 12G 입니다!





1편 라이캣의 탄생

1. 캣의 일상
2. 캣의 정보 변경
3. 각 변수들의 타입을 알아보기
4. 각 변수들의 속성을 알아보기
 - 4.1 int의 속성 알아보기
 - 4.2 float의 속성 알아보기
 - 4.3 str의 속성 알아보기
 - 4.4 bool의 속성 알아보기
 - 4.5 list, tuple의 속성 알아보기
 - 4.6 dict의 속성 알아보기
 - 4.7 set의 속성 알아보기
 - 4.8 list, tuple, set, dict에 대해 더 알아 보기
5. 각 변수들을 형변환 해보기
6. 출력을 하는 여러가지 용법

1. 캣의 일상

한편, 유니브 월드 외곽에서는 생선가게를 운영하는 평민 '캣'이 살고 있었다.



생선 가게를 운영하는 평범한 소년 '캣'이 있습니다. 이 소년은 아픈 어머니를 대신하여 새벽이면 험한 바다에 나가 고기를 잡고, 오후가 되면 생선을 파는 생활을 반복하였어요.



캣에게는 비밀이 한가지 있었어요. 부모님께서 신신당부하여 남들에게 말하지 않았지만, 자신이 원할 때 자신의 능력치를 볼수 있는 신기하고 특별한 능력이 있었어요.

"나에겐 **고기잡기**와 **고기팔기 스킬**이 있다냥, 매일매일 훈련해서 이 두개의 스킬 만큼은 최고의 스킬로 만들어보자냥!"

캣은 묵묵히, 매일 실력을 갈고닦았습니다.

```

#[In]

# 라이캣의 개인정보 입력하기

이름 = '캣'
설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
나이 = 10
오늘_잡은_물고기 = '10'
키 = '45cm'
몸무게 = 1.2
육식 = True
초식 = True
돈 = 1000
혼장 = []
기술 = ['고기잡이', '고기팔기']

...

라이캣의 개인정보입니다.
성장함에 따라 해당 값이 변합니다.

주의) 마음대로 성장시키지 마십시오.
...

```

Python ▾

이름, 나이, 오늘_잡은_물고기 등을 변수라고 합니다. 변수는 변할 수 있는 수입니다. 또, 맨 위에 있는 # 으로 되어 있는 부분과 ... 로 감싸여져 있는 부분은 주석이라고 해요. Code의 양이 많거나 복잡하여 짧은 시간 내 이해하기 힘들 때 이해를 돕기 위해 주석이 필요합니다. 이 부분은 실행되지 않습니다. 코드를 잠시 보류하는 용도로도 사용하죠.

이렇게 하고 **Alt + Enter** 를 실행해 보세요. 아래 셀이 추가되었죠? 따로 메시지는 뜨지 않았을 거예요. 무언가 실행시키지 않았기 때문입니다. 실행은 그 셀만 실행하는 **Ctrl + Enter** 와 실행하고 셀 아래 셀을 추가하는 **Alt + Enter** 두가지가 있습니다.

 주석으로 변경을 원하는 코드에서 **Ctrl + /** 를 누르면 주석처리 됩니다. 여러줄도 가능해요.

2. 캣의 정보 변경

```
#[In]
# 라이캣의 개인정보 변경하기

나이 = 11
이름, 나이
```

Python ▾

이렇게 하고 **Alt + Enter** 를 실행해 보세요. 나이에 11이 출력되었죠? 이처럼 변수는 변경할 수 있습니다. 그리고, 위에 있는 변수를 가져와 사용할 수도 있어요.

```
#[In]

나이 = 나이 + 1
나이
```

Python ▾


연산은 나중에 할 예정이지만, 이렇게 덧셈을 할 수도 있습니다. 그럼 아래와 같이 덧셈을 해볼게요.

```
#[In]

# 오늘_잡은_물고기 = '10'
오늘_잡은_물고기 = 오늘_잡은_물고기 + 1
```

Python ▾

이렇게 연산을 실행하게 되면 Error 문구가 뜨게 됩니다. 그 이유는 오늘 잡은 물고기는 문자열이기 때문인데요. 이처럼 파이썬과 같은 **고급프로그래밍 언어**에서는 그 형태를 구분하여 걸맞는 연산이 가능하도록 장치를 해두었습니다.

 Python이 무엇이고, 무엇을 할 수 있는지는 아래 영상을 참고해주세요. 고급 프로그래밍 언어가 어떤 뜻인지도 설명하고 있습니다. (책으로 보시는 분들은 YouTube에서 제주코딩베이스캠프를 검색해주세요.)

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





가능하다면, 1편인 Front-end도 시청해주세요. 😊

조금 더 알려드리자면, 변수에는 몇 가지 규칙이 있어요.

1. 변수 중간에 띄어쓰기를 하지 않습니다. 띄어쓰기를 명시하고 싶으실 때에는 언더스코어(`_`)를 사용하세요!
2. 첫 글자는 숫자나 특수문자를 쓰지 않습니다. 물론 언더스코어(`_`)제외입니다.
3. 첫 글자를 대문자로 쓰지 않습니다.(Class가 대문자로 쓰기 때문이지만, 대문자로 써도 실행됩니다.)
4. 예약어를 사용하지 않습니다. 예를 들어 뒤에 `print` 구문이 나오는데요. 이러한 함수명이나 구문은 변수명으로 사용하지 않습니다.
5. 변수명은 대소문자를 가립니다! `Apple` 과 `apple` 은 다른 변수가 됩니다.

```
#[In]

# 오늘_잡은_물고기 = '10'
오늘_잡은_물고기 = int(오늘_잡은_물고기) + 1
오늘_잡은_물고기
```

Python ▾

자, 위와 같이 변경하고 `Alt + Enter` 를 눌러 실행시켜보세요! 연산이 되었죠? 그 이유는, `int` 라는 형변환 함수가 문자열을 숫자로 바꾸어 주었기 때문입니다. 형변환은 이번 챕터 5장에서 자세히 살펴볼 것입니다.

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. 각 변수들의 타입을 알아보기

이번에는 위에서 코드를 복사해와서 아래와 같이 작성 후 실행해보세요.

```
#[In]

print(type(이름)) # '켓'
print(type(나이)) # 현재 12
print(type(오늘_잡은_물고기)) #현재 11
print(type(몸무게))
print(type(육식)) #True
print(type(기술)) #['고기잡이', '고기팔기']
```

Python ▾



`print` 를 써도 되고 쓰지 않아도 되는 것인가요? 네, 주피터 노트북에서는 `print` 를 쓰지 않아도 마지막에 있는 변수는 출력을 합니다.(Python 기본 에디터 사용시 Error!) 그러나 여러개를 출력할 때에는 꼭 `print` 를 명시해주셔야 합니다.

4. 각 변수들의 속성을 알아보기

위에서 실행시킨 결과값은 아래 주석과 같이 나왔을거예요! 물론 더 많은 자료형이 있지만, 이 수업은 기초 수업이기 때문에 간단한 내용들만 살펴볼 것입니다.

```
#[In]

print(type(이름)) # class 'str' - 문자열
print(type(나이)) # class 'int' - 정수
print(type(오늘_잡은_물고기)) # class 'int' - 정수
print(type(몸무게)) # class 'float'은 실수입니다.
print(type(육식)) # class 'bool' - 참거짓
print(type(기술)) # class 'list' - 배열
```

Python ▾

각 변수들에는 속성이 있습니다. 예를 들어 스트링의 경우에는 덧셈을 하면 이어 붙인다든지 하는 속성이요.

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



```
#[In]

x = '10'
y = 10
print(x + x)
print(y + y)
```

Python ▾

위와 같이 실행을 시켰을 경우 `'10' + '10'` 은 `'1010'` 이 되지만, `10 + 10` 은 `20` 이 됩니다. 이렇게 각 자료형마다 특징들이 있고, 이번 챕터에서는 이 특징들을 살펴볼 생각이예요. 그러나 걱정마세요. 너무 깊게 다루지는 않을 것입니다.



Q: 깊게 다루지 않고 데이터 분석, 서비스 개발, IoT, 게임 개발 등을 할 수 있나요?

A: 아닙니다! 하지만, 초급자 분들은 문법에 매몰 되시면 안되요. 일단 한 두 서클을 돌아보시면, 자연스럽게 이해되는 문법들이 있고, 또 그렇게 한 번 성공한 결과물을 갖게되면 자신감도 생기거든요! 그러니 초급자 분들은 한 서클(결과물을 만드는 과정)을 도시는 것에 초점을 맞춰서 공부해주세요!

4.1 int의 속성 알아보기

int는 정수입니다! 정수가 가진 속성에 대해 알아보도록 하겠습니다. 우선 아래와 같이 실행시킨 다음에 말씀드리도록 하겠습니다.

```
#[In]

나이 = 10
print(type(나이))
print(dir(나이))
```

Python ▾

그럼 아래와 같이 무시무시한 길이의 코드가 나타납니다. 겁내지 마세요. 다 알 필요도 없을 뿐더러, 우리는 아주 알고 넓게, 실용적인 부분만 배울 것이기 때문입니다.

일단 첫줄 먼저 설명해드릴게요. `<class 'int'>`, class는 맨 뒤에 챕터에서 다루게 되고, int는 우리가 앞에서 공부한 것처럼 정수형을 나타냅니다. 이처럼 `type(변수)`를 해보시면 원하는 변수의 타입을 알아낼 수 있어요.

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



```
#[Out]

<class 'int'>
['__abs__', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__', '__divmod__', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__', '__getattr__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__init_subclass__', '__int__', '__invert__', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__', '__or__', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__', '__repr__', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__', '__rrshift__', '__rshift__', '__rsub__', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__', '__sub__', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

Python ▾

여러분, 아래와 같이 정수와 실수를 더하면 얼마가 나올까요? 네, **20.1** 입니다. 그런데 다른 언어들은 이렇게 융통성이 있지 않아요. 만약 이 연산이 다른 언어였다면(JS는 더 융통성이 있으니 제외입니다.) 아래와 같이 말했을 거예요.

```
#[In]

#나이는 10이고, 몸무게는 10.1 입니다.

나이 + 몸무게
```

Python ▾

실수로 더할꺼니? 그럼 실수로 바꿔줘야지. 정수로 더할꺼니? 그럼 뒤에 수에서 0.1을 포기해.

그런데 파이썬은 놀랍게도(!?), 이 연산이 가능합니다. 위 처럼 실행을 시켰다면 **20.1**의 출력 결과가 나올게요. 그럼 아래와 같은 코드는 어떨까요?

```
#[In]

10 + '10'
```

Python ▾

이것을 실행시키면 또 아래와 같은 무시무시한 예러가 나옵니다. 앞으로 자주 만나게 될 예러명이기 때문에 한번 읽어보죠.

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



```
#[Out]
...
TypeError: unsupported operand type(s) for +: 'int' and 'str'
```

Python ▾

타입에러인데 정수형(int)와 문자열(str)의 덧셈을 허락하지 않는다고 합니다. 자, 이렇게 덧셈에 대해 어떻게 할 것인지에 대해 정의되어 있는 부분이, `'_add_'`입니다! 이부분은 class 챗터에서 더 배우게 되실거예요. 곱하기는 `'_mul_'`, 비교는 `'_eq_'`합니다.

아, 언더바(_)는 하나가 아니라 두개예요! 그래서 던더함수라고 부르기도 하죠. 매직메서드라고 부르는 것이 통상적입니다.

그렇다면, 언더바가 없는 것들의 정체는 무엇일까요? 아래와 같은 코드를 실행해볼게요.

```
#[In]

나이 = 10
print(나이.bit_length())
print(bin(나이))

나이_90년후 = 100
print(나이_90년후.bit_length())
print(bin(나이_90년후))
```

Python ▾

이것들은 `.`을 찍어 사용할 수 있어요. `bit_length()`만 사용해보았습니다. 이 **메서드**(지금은 점을 찍어서 사용할 수 있는 코드라고만 이해를 해주세요.)는 각 숫자를 bit로 변환한 자리숫자를 출력해준답니다! bit는 0과 1로 이루어진 세계입니다. **2진법**을 사용하죠.

자, 너무 많은 내용을 했어요. 모두 이해하실 필요 없습니다. 여기서 중요한 핵심은 `.`을 찍어 활용할 수 있는 코드를 `dir`로 볼 수 있다는 사실만 알고 넘어가도록 하겠습니다!

4.2 float의 속성 알아보기

float형은 실수입니다! 실수가 있으니 허수도 있을까요? 네, 물론입니다. 하지만, 우리 수업에서 그렇게 어려운 수학적인 내용은 다루지 않아요. 아래와 같이 실행해본 후 이 챗터는 넘어갑시다.

```
#[In]

# 몸무게는 10.1kg 입니다!
print(type(몸무게))
print(dir(몸무게))
```

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



4.3 str의 속성 알아보기

`str` 자료형은 스트링이라 읽습니다. 이 자료형은 작은따옴표(' ')나 큰따옴표(" ") 또는 삼중따옴표(""" """)로 둘러싸서 나타냅니다. 삼중따옴표를 사용할 경우에는 줄단위의 문자열을 나타낼 수 있습니다. 이러한 문자열은 **시퀀스 자료형**에 해당됩니다.

Sequence, 순서가 있는 자료형이라는 뜻으로 각 요소들은 해당하는 **Index 값을 갖습니다**. 파이썬에서 **가장 많이 사용되는 형태의 자료형**이며, 시퀀스 자료형이 가지는 특성을 통하여 인덱싱, 슬라이싱, 반복 등 여러 연산으로 그 활용 범위가 넓습니다. 파이썬에서는 **문자열, 리스트** 등이 시퀀스 자료형에 해당됩니다.

```
#[In]

# 설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
# 기술 = ['고기잡이', '고기팔기']

print(설명[0])
print(설명 [1])
print(설명 [2])
print(기술 [0])
print(기술 [1])
print(기술 [0][0])
print(기술 [0][1])
```

Python ▾

```
#[Out]

위
니
브
고기잡이
고기팔기
고
기
```

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 `설명[0]` 안에 들어가는 0이라는 숫자를 `index`라고 부릅니다! 그리고 이렇게 `index`를 활용하여 특정 값을 호출하는 방식을 `indexing`이라고 하죠!

또, 이 `index`를 활용하여 문자열에서 원하는 부분만을 추출해낼 수 있습니다.

```
#[In]

설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
print(설명[0:6])
```

Python ▾



인덱스가 `start` 인 지점에서 `end` 미만인 지점까지 추출합니다. 여기서 `start` 를 생략했을 경우 문자열의 시작 지점, `end` 를 생략했을 경우 문자열의 끝지점이 설정됩니다.

```
#[In]

캣의_생년월일 = "2220.02.22"
생년 = birth[:4]
월 = birth[5:7]
일 = birth[8:]
print("태어난 연도 : ", 생년)
print("태어난 월 : ", 월)
print("태어난 일 : ", 일)
```

Python ▾

```
#[Out]

태어난 연도 : 2220
태어난 월 : 02
태어난 일 : 22
```

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



변수명[start:stop:step] 와 같은 형식으로 사용도 가능합니다. 이 경우 순회가능한 객체에 start index 부터 stop index 바로 전만큼의 범위에서 k만큼 건너뛰게 됩니다.

```
#[In]
```

```
숫자 = '123456789101112'  
print(숫자[::-1])  
print(숫자[1:7:2])
```

Python ▾

```
#[Out]
```

```
'211101987654321'  
'246'
```

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



4.4 bool의 속성 알아보기

`bool` 자료형은 부울형, 불리언 자료형이라 읽습니다. 이 자료형은 `True`와 `False` 2가지 타입밖에 없어요.

💡 javascript에서는 `True`를 소문자 형태인 `true`로 표시하고 있어요! 이런 차이점에 대해서 알고 있으시면 좋습니다.

```
#[In]

육식 = True
초식 = True
print(type(육식))
print(dir(초식))
```

Python ▾

```
#[Out]

<class 'bool'>
['_abs_', '_add_', '_and_', '_bool_', '_ceil_', '_class_', '_delattr_', '_dir_', '_divmod_', '_doc_', '_eq_', '_float_', '_floor_', '_floordiv_', '_format_', '_ge_', '_getattr_', '_getnewargs_', '_gt_', '_hash_', '_index_', '_init_', '_init_subclass_', '_int_', '_invert_', '_le_', '_lshift_', '_lt_', '_mod_', '_mul_', '_ne_', '_neg_', '_new_', '_or_', '_pos_', '_pow_', '_radd_', '_rand_', '_rdivmod_', '_reduce_', '_reduce_ex_', '_repr_', '_rfloordiv_', '_rlshift_', '_rmod_', '_rmul_', '_ror_', '_round_', '_rpow_', '_rrshift_', '_rshift_', '_rsub_', '_rtruediv_', '_rxor_', '_setattr_', '_sizeof_', '_str_', '_sub_', '_subclasshook_', '_truediv_', '_trunc_', '_xor_', 'bit_length', 'conjugate', 'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

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



4.5 list, tuple의 속성 알아보기

`list` 는 두 개 이상의 값을 저장하고 싶을 때 사용하는 자료형입니다. `str` 처럼 순서가 있는 시퀀스 자료형이며, 각 원소들은 변경이 가능합니다. 리스트의 기본 형태는 아래와 같습니다.

```
#[In]

훈장 = []
기술 = ['고기잡이', '고기팔기']
print(type(기술))
print(dir(기술))
```

Python ▾

```
#[Out]

<class 'list'>
['_add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__', '__getattr__', '__getitem__', '__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__init_subclass__', '__iter__', '__le__', '__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__', '__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear', 'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

Python ▾

여기서 대괄호 (`[]`)를 사용하지 않고 소괄호 (`()`)를 사용하면 `list`가 아니라, 자료의 값이 변경 불가능한 `tuple` 이 됩니다. `tuple`은 순서가 있고, 소괄호를 사용하며, 값을 변경할 수 없습니다. 알은 물에서는 튜플을 상세하게 다루지 않으니 가볍게 훑고 넘어가주세요.

```
#[In]

잡은물고기_튜플 = ('광어', '고등어', '오징어', '오징어', '광어', '광어', '고등어', '고등어', '백상아리', '금붕어')
```

Python ▾

앞서 말씀드린 것처럼 `list` 는 순서가 있는 시퀀스형 자료형이기 때문에 아래와 같이 `str` 처럼 `indexing, slicing`이 가능합니다.

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



#[In]

```
잡은물고기 = ['광어', '고등어', '오징어', '오징어', '광어', '광어', '고등어', '고등어', '백상아리', '금붕어']
```

Python ▾

#[In]

```
print(잡은물고기[0])
print(잡은물고기[0:3])
print(잡은물고기[0:7:2])
```

Python ▾

#[Out]

```
광어
['광어', '고등어', '오징어']
['광어', '오징어', '광어', '고등어']
```

Python ▾

그런데 마지막에 잡은 물고기가 금붕어죠? 잘못 표기한 것은 아래와 같이 바꿀 수 있습니다.

#[In]

```
잡은물고기[-1] = '백상아리' # -1 인덱스는 마지막에 있는 값으로, 아래와 같은 의미를 지닙니다.
잡은물고기[9] = '백상아리'
```

잡은물고기

Python ▾

#[Out]

```
['광어', '고등어', '오징어', '오징어', '광어', '광어', '고등어', '고등어', '백상아리', '백상아리']
```

Python ▾

여기서 광어는 얼마나 잡았는지, 고등어는 얼마나 잡았는지 알고 싶으면 어떻게 할까요? 여기서 앞서 말씀드린 **메서드**를 사용할 수 있습니다. 아래 코드를 실행해 보세요. **메서드 정리는 다른 강의에서** 해드릴거예요. 여기는 얇은물 강좌이니, 어떻게 사용하는지 가볍게 살펴봅시다.

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



```
#[In]
잡은물고기.count('백상아리')
```

Python ▾

```
#[Out]
```

```
2
```

Python ▾

대단하군요. 고양이가 백상아리를 잡다니요. 그럼 훈장에 '백상아리를 잡은 고양이'를 추가해볼게요. 이 역시 메서드를 사용합니다.

```
#[In]
훈장.append('백상아리를 잡은 고양이')
훈장
```

Python ▾

```
#[Out]
```

```
'백상아리를 잡은 고양이'
```

Python ▾

4.6 dict의 속성 알아보기

그런데 이렇게 물고기를 저장하는 것은 비효율적이겠죠? 우리는 '백상아리!' 하면 '2마리!' 이렇게 바로 알려주었으면 좋겠는데요. 이것이 가능한 자료형이 Dictionary입니다.

```
# Dictionary의 구조
dic = { 'key' : 'value' }
```

Python ▾

자, 그러면 위에 리스트로 저장되어 있던 물고기를 딕셔너리로 정리해봅시다.

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



```
#[In]
```

```
잡은물고기_딕셔너리 = {'광어' : 3, '고등어' : 2, '오징어' : 2, '백상아리' : 2}
```

Python ▾

어떤가요? 보다 간편해졌죠? 우리는 앞으로 아래와 같이 key 값으로 value를 호출할 수 있습니다.

```
#[In]
```

```
잡은물고기_딕셔너리['광어']
```

Python ▾

```
#[Out]
```

```
2
```

Python ▾

값을 추가하고 싶을 때에는 아래와 같은 방법을 사용할 수 있습니다.

```
#[In]
```

```
잡은물고기_딕셔너리['고래'] = 1  
잡은물고기_딕셔너리
```

Python ▾

```
#[Out]
```

```
{'광어' : 3, '고등어' : 2, '오징어' : 2, '백상아리' : 2, '고래' : 1}
```

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



값을 지울 때에는 아래와 같은 방법을 사용합니다.

```
#[In]

del 잡은물고기_딕셔너리['고래']
#del 키워드는 다른 자료형에서도 데이터를 지울 때 사용할 수 있습니다.

잡은물고기_딕셔너리
```

Python ▾

```
#[Out]

{'광어' : 3, '고등어' : 2, '오징어' : 2, '백상아리' : 2}
```

Python ▾

Key값만 따로 알고 싶을 때와 Value값만 따로 알고 싶을 때에는 아래와 같은 방법을 사용합니다.

```
#[In]

print(잡은물고기_딕셔너리.keys())
print(잡은물고기_딕셔너리.values())
print(잡은물고기_딕셔너리.items())
```

Python ▾

```
#[Out]

dict_keys(['광어', '고등어', '오징어', '백상아리'])
dict_values([3, 2, 2, 2])
dict_items([('광어', 3), ('고등어', 2), ('오징어', 2), ('백상아리', 2)])
```

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



4.7 set의 속성 알아보기

set은 집합이에요. 중복을 허락하지 않죠! 혹시 차집합, 합집합, 교집합 아시나요? 그 집합입니다! 자, 그럼 어떻게 사용할까요?

```
#[In]

잡은물고기_집합 = set(잡은물고기)
잡은물고기_집합
```

Python ▾

```
#[Out]

잡은물고기 = {'광어', '고등어', '오징어', '백상아리'}
```

Python ▾

어떨까요? 중복이 사라졌습니다! 이제 잡은 물고기의 종류를 한 번에 볼 수 있죠!

4.8 list, tuple, set, dict에 대해 더 알아보기

자료형끼리의 사칙연산과 메서드는 한 번 정리할 필요가 있어요. 더 공부하기를 원하신다면, 아래 영상을 참고하여 각 자료형 옆에 있는 메모 노트에 메모해 주세요. 하지만 기억하세요. 먼저 빠르게 훑어 간단한 웹이나 앱, 게임이나 IoT와 같은 결과물을 만들고, 나중에 좀 더 깊게 파봐도 늦지 않습니다. 너무 많은 내용을 한꺼번에 담으려 하지 마세요.



Youtube '제주코딩베이스캠프'의 영상입니다 😊

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. 각 변수들을 형 변환 해보기

자, 형 변환에 대해 좀 더 깊게 알아보시다.

```
#[In]

print(type(int(3.5)))
print(int(3.5))
print(type(float(3)))
print(float(3))
print(type(str(3)))
print(str(3))
```

Python ▾

```
#[Out]

<class 'int'>
3
<class 'float'>
3.0
<class 'str'>
3
```

Python ▾

앞서 말씀드린 것처럼 기존에 자료형에서 다른 자료형으로 바꾸는 것을 형 변환이라고 합니다. 아래 처럼 `input` 함수를 이용하면 숫자나 문자를 입력받을 수 있는데요. 둘 다 `str` 로 변수를 받기 때문에 주의해서 사용을 해야 합니다.

```
#[In]

x = input('좋아하는 숫자를 입력하세요 :')
y = input('더할 숫자를 입력하세요 :')
print(x + y)
print(type(x))
print(type(y))
```

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



```
#[Out]

좋아하는 숫자를 입력하세요 : 123
더할 숫자를 입력하세요 : 123
123123
<class 'str'>
<class 'str'>
```

Python ▾

파이썬에서의 형 변환은 아주 많은 방법들이 있지만 **내장함수(built in-functions, 이 키워드는 매우 중요하니 꼭 암기해주세요.)**를 이용해서 아주 쉽게 할 수 있습니다. 어떤식으로 쓰이는지 간단한 예시를 통해 같이 한번 살펴봅시다.

형변환

- 🌟 [int](#) 다른 자료형을 정수형으로 변환
- 🌟 [str](#) 다른 자료형을 문자열로 변환
- 🌟 [float](#) 다른 자료형을 실수형으로 변환
- 🌟 [list](#) 다른 자료형을 리스트로 변환
- 🌟 [tuple](#) 다른 자료형을 튜플형으로 변환
- 🌟 [set](#) 다른 자료형을 집합 자료형으로 변환
- 🌟 [dict](#) 다른 자료형을 사전형으로 변환
- + [New](#)

- [int로 형변환](#)

```
#[In]

오늘잡은_물고기_수 = '371'
print(type(오늘잡은_물고기_수))
print(type(int(오늘잡은_물고기_수))
print(int(오늘잡은_물고기_수))
```

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



```
#[Out]

<class 'str'>
<class 'int'>
371
```

Python ▾

- string으로 형변환

```
#[In]

오늘잡은_물고기_수 = 371
print(type(오늘잡은_물고기_수))
print(type(str(오늘잡은_물고기_수)))
print(str(오늘잡은_물고기_수))
```

Python ▾

```
#[Out]

<class 'int'>
<class 'str'>
'371'
```

Python ▾

- bool형으로 형변환

```
#[In]

print("bool('test') : ", bool('test!!'))
print("bool(1) : ", bool(1))
print("bool(0) : ", bool(0))
print("bool(-1) : ", bool(-1))
print("bool(' ') : ", bool(' '))
print("bool('') : ", bool(''))
print("bool(None) : ", bool(None))
```

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



```
#[Out]


bool('test') : True
bool(1) : True
bool(0) : False
bool(-1) : True
bool(' ') : True
bool('') : False
bool(None) : False
```

Python ▾

bool()함수는 **인자값**(아규먼트, argument, parameter와는 차이가 있으니 함수에서 정리해드리도록 하겠습니다.)을 Boolean 자료형으로 형변환하게 됩니다. 부울 값은 True와 False로 나뉩니다.

부울 값은 **직접 입력된 값일 수도 있고 부울 연산에 의해 나온 결과값일 수도 있습니다.**

예를 들어 $x = 10$ 일 때 $x > 100$ 은 False이죠. 또한 이미 Python 내에서 규정한 부울 값일 수도 있습니다. 예를 들어 0은 False, 0을 제외한 다른 숫자는 True입니다.

 list, tuple, dict, set은 형변환을 어떻게 할까요? 옆 노트에 정리해보세요.

6. 출력을 하는 여러가지 용법

`print` 내장함수(built-in functions, 빌트인펑션)를 사용하여 출력하는 방법에는 여러가지 방법이 있습니다. 물론 우리가 사용하는 `colab` 이나 `jupyter notebook` 은 마지막 라인에 한하여 `print` 를 쓰지 않아도 출력합니다.

```
print('1. 제 이름은 ', 이름, '입니다. 제 나이는 ', 나이, '입니다')
print(f'2. 제 이름은 {이름}입니다. 제 나이는 {나이}입니다.')
print('3. 제 이름은 {}입니다. 제 나이는 {}입니다.'.format(이름, 나이))
print('4. 제 이름은 %s입니다. 제 나이는 %d입니다.'%(이름, 나이))
```

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 비교적 최근에 나온 2번 문법이 자주 사용됩니다. 4번은 잘 사용되지 않으니 참고바랍니다.

1. ,(콤마) 로 연결하는 방법입니다. 콤마의 단위는 오브젝트입니다. 이름, 나이 모두 오브젝트의 한 단위이고 '1. 제 이름은', '입니다. 제 나이는' 등의 문자열도 모두 오브젝트입니다. 앞에서는 변수라고 배웠죠? 변수에 넣을 수 있는 모든 값 또는 변수도 오브젝트입니다. 예를 들어 아래와 같이 변수나 그 값을 직접 넣는 것은 같은 값을 출력합니다.

```

힘 = 12
print('제 힘은 ', 힘, '입니다.')
print('제 힘은 ', 12, '입니다.')
    
```

Python ▾

2. python 3.6 version 이상에서는 f-string 용법을 지원합니다. 사용하는 방식은 {}(중괄호) 에 변수 이름을 넣으시면 됩니다.
3. format을 이용한 문자열 포매팅 방식입니다. 앞서 dir('문자열') 에서 던더함수 뒤에 있는 메서드를 확인해보시면 format 메서드가 있습니다. {}(중괄호) 로 변수의 자리를 비워놓고 '문자열'.format() 괄호 안에 넣고 싶은 오브젝트를 넣어주는 방법입니다. 역시 변수는 콤마로 구별해 넣습니다.
4. 잘 사용하지 않는 방법입니다. 포맷코드를 이용한 문자열 포매팅입니다. 넣고 싶은 오브젝트는 % 뒤에 넣습니다. 넣을 값에 맞춰서 알맞은 자료형에 대한 포맷코드를 선택해야 합니다. 아래 코드 표는 참고삼아 훑고 넘어가세요.

포맷 코드의 종류

★ %c	문자
★ %s	문자열
★ %d	정수
★ %f	실수
★ %b	2진수
★ %o	8진수
★ %x	16진수
+ New	

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





2편 생선을 잡아라

1. 썬의 다집
2. 썬의 생선팔기
3. 대입 연산자(할당 연산자)
4. 비교 연산자
5. 논리 연산자
6. 비트 연산자
7. 멤버 연산자
8. 식별 연산자
9. 연산자의 우선순위

1. 캣의 다짐



어느날 생선가게를 운영하며 열심히 살아가던 캣에게 슬픔이 찾아왔습니다. 아프던 캣의 어머니의 병세가 더 위독해진 것이었어요.

당장 병원에 가야했지만 '위니브 월드'에서의 병원은 왕족인 사자와 부자에게만 허락된 공간이었습니다. 평민이었던 캣의 어머니는 병원에 가지 못하였고 캣에게 생선가게를 부탁하였습니다.

"생선가게를 부탁하마..."

"엄마.."



캣은 다짐했습니다.

▮ 평민도 이용할 수 있는 병원을 세울꺼다냥!!

캣은 병원을 세우기 위해 자신의 스킬을 활용하여 더 열심히 생선을 잡고 팔았습니다.

• 라이캣의 현재 상태창!

```
#[In]

이름 = '캣'
설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
나이 = 13
오늘_잡은_물고기 = '10'
키 = '45cm'
몸무게 = 1.2
육식 = True
초식 = True
돈 = 5000
훈장 = ['백상아리를 잡은 고양이']
기술 = ['고기잡이', '고기팔기']
```

Python ▾

캣은 장사를 할 때 캣만의 규칙이 있었습니다. 잡은 물고기를 당일 다 팔아야 하고 다음날 매출은 오늘 매출의 2배로 목표를 잡습니다.

```
#[In]

# 캣의 규칙
오늘_잡은_물고기 = 0
오늘_판_물고기 = 0
매출 = 0

# 물고기를 10마리 잡았습니다.
# 모든 물고기는 100원입니다.

오늘_잡은_물고기 = 오늘_잡은_물고기 + 10
오늘_판_물고기 = 오늘_잡은_물고기
재고 = 오늘_잡은_물고기 - 오늘_판_물고기
매출 = 100*10

#f-string 출력 방법을 사용해서도 좋습니다.
print('나는 오늘', 오늘_잡은_물고기, '마리를 잡았고', 오늘_판_물고기, '마리를 팔았다냥')
print('재고는', 재고, '마리다냥!')
print('오늘', 매출, '원을 벌었으니 내일은', 매출*2, '원을 벌테다!')
```

Python ▾

오늘_잡은_물고기 + 10 , 오늘_판_물고기 = 오늘_잡은_물고기 , 오늘_잡은_물고기 - 오늘_판_물고기 , 100*10 에서 볼 수 있는 +, =, -, * 와 같이 값을 계산하거나 대입하는 것을 연산자라고 합니다.

연산자의 종류로는 산술 연산자, 대입 연산자(할당연산자), 비교 연산자, 논리 연산자, 비트 연산자, 멤버 연산자, 식별 연산자가 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



💡 종류가 많기 때문에 어려움을 느끼실 수 있지만 이 연산자들을 한번에 알려고 하는 것은 큰 의미가 없습니다. 필요할 때 사용해 보면서 익히는 것이 중요합니다.

2. 캣의 생선 팔기

캣은 매출을 올리기 위해 모든 물고기를 같은 가격이 아닌 등급에 따라 다르게 책정하기로 합니다.

물고기 등급에 따라 가격을 다르게 팔아볼까냥?

```
#[In]

# A등급 : 1000원, B등급 : 500원, C등급 : 100원
# 오늘 잡은 물고기 : A등급 5마리, B등급 7마리, C등급 10마리

A등급 = 5
B등급 = 7
C등급 = 10
매출 = 0
```

Python ▾

등급에 따라 가격을 책정하여 장사를 시작한 캣의 생선 가게에 손님이 찾아왔습니다.

```
#[In]

# 손님의 주문
# A등급 2마리, B등급 3마리 주세요.
# 받은 돈은 4000원 입니다.

A등급 = A등급 - 2
B등급 = B등급 - 3

합계 = 1000*2 + 500*3
받은_돈 = 4000

print(f'감사합니다. 여기 거스름 돈 {받은_돈 - 합계}원 입니다.')

재고 = A등급 + B등급 + C등급
매출 = 매출 + 합계

print(f'현재 매출 : {매출}, 재고 : {재고}')
```

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 눌러주면 아래와 같이 출력됩니다.

💡 물고기의 등급과 가격을 dict로 구현해보시고, 계산해보세요!

#[Out]

감사합니다. 여기 거스름돈 500 원입니다.
현재 매출 : 3500 재고 : 17

Python ▾

오늘도 잡은 물고기를 다 팔아 재고를 0으로 장사를 마무리 하는 켓. 하지만 유니브 월드에서는 당일 매출의 4분의 1을 세금으로 납부해야 한다고 합니다.

예휴 오늘도 세금을 납부해야겠구냥... 오늘은 또 얼마를 내야하냐?

#[In]

```
# 세금 납부하기
총매출 = 10000
세금 = 총매출/4
순이익 = 총매출-세금

print(총매출, 세금, 순이익)
# 10000 2500.0 7500.0
```

Python ▾

`+`, `-`, `*`, `/` 와 같은 연산자를 산술연산자라고 합니다. 산술연산자는 가장 많이 접할 수 있고, 자주 사용되는 연산자입니다. 사칙연산과 더불어 제곱, 나머지 연산자 등이 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



산술 연산자

Aa 기호	이름	설명	예제
+	덧셈	연산자의 왼쪽과 오른쪽 값을 더합니다.	a + b
-	뺄셈	연산자의 왼쪽 값에서 오른쪽 값을 뺍니다.	a - b
*	곱셈	연산자의 왼쪽 값과 오른쪽 값을 곱합니다.	a * b
**	제곱	연산자의 왼쪽 값에서 오른쪽 값만큼 제곱합니다.	a ** b
/	나눗셈	연산자의 왼쪽 값을 오른쪽 값으로 나눕니다.	a / b
//	몫	연산자의 왼쪽 값을 오른쪽 값으로 나눈 몫을 구합니다. (따라서 결과는 정수)	a // b
%	나머지	연산자의 왼쪽 값을 오른쪽 값으로 나눈 나머지를 구합니다.	a % b

COUNT 7

```
#[In]

a = 5
b = 2

print('a + b = ', a + b) # 7
print('a - b = ', a - b) # 3
print('a * b = ', a * b) # 10
print('a / b = ', a / b) # 2.5
print('a ** b = ', a ** b) # 25
print('a // b = ', a // b) # 2
print('a % b = ', a % b) # 1
```

Python ▾

💡 다른 프로그래밍 언어에서 연산자를 이미 학습하신 분들은 눈치채셨을 수도 있으시겠지만, 파이썬에는 전/후위 연산자(++, --)가 존재하지 않습니다!

자, 성실히 세금을 납부했으니 훈장을 부여하도록 합시다.

```
#[In]

훈장.append('성실한 납세자')
```

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. 대입 연산자(할당 연산자)

대입 연산자는 산술 연산의 코드를 축약해서 사용할 수 있도록 지원하는 기능입니다.

피연산자를 한번만 사용하기 때문에 대입 연산자를 잘 활용한다면 코드를 좀더 간결하게 사용할 수 있다는 장점이 있습니다.

위의 예제에서 사용했던 코드를 가져와서 알아보시다.

```
#[In]

# 1번 예제
오늘_잡은_물고기 = 오늘_잡은_물고기 + 10
오늘_판_물고기 = 오늘_잡은_물고기
```

Python ▾

```
#[In]

# 대입 연산자
오늘_잡은_물고기 += 10
오늘_판_물고기 = 오늘_잡은_물고기
```

Python ▾

`오늘_판_물고기 = 오늘_잡은_물고기` 는 대입 연산자로 오른쪽 값을 왼쪽 변수에 할당하였습니다.

`오늘_잡은_물고기 = 오늘_잡은_물고기 + 10` 는 덧셈 대입을 사용하여 `오늘_잡은_물고기 += 10` 로 축약하여 사용할 수 있고 연산 결과는 동일합니다.

다른 산술 연산자도 모두 대입 연산자로 축약하여 사용할 수 있습니다.

```
#[In]

a = 10
b = 2

a += b # 12
a -= b # 10
a *= b # 20
a /= b # 10.0
a **= b # 100.0
a //= b # 50.0
a %= b # 0.0
```

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



대입 연산자(할당 연산자)

Aa 기호	이름	설명	예제	동일
=	대입	연산자의 오른쪽 값을 왼쪽 변수에 할당합니다.	a = b	a = b
+=	덧셈 대입	연산자의 왼쪽 변수의 값과 오른쪽 값을 더한 결과를 왼쪽 변수에 할당합니다.	a += b	a = a + b
-=	뺄셈 대입	연산자의 왼쪽 변수의 값에서 오른쪽 값을 뺀 결과를 왼쪽 변수에 할당합니다.	a -= b	a = a - b
*=	곱셈 대입	연산자의 왼쪽 변수의 값과 오른쪽 값을 곱한 결과를 왼쪽 변수에 할당합니다.	a *= b	a = a * b
**=	제곱 대입	연산자의 왼쪽 변수의 값에서 오른쪽 값만큼 제곱한 결과를 왼쪽 변수에 할당합니다.	a **= b	a = a ** b
/=	나눗셈 대입	연산자의 오른쪽 변수의 값을 오른쪽 값만큼 나눈 결과를 왼쪽 변수에 할당합니다.	a /= b	a = a / b
//=	몫 대입	연산자의 왼쪽 변수의 값을 오른쪽 값만큼 나눈 몫을 왼쪽 변수에 할당합니다.	a //= b	a = a // b
%=	나머지 연산 대입	연산자의 왼쪽 변수의 값을 오른쪽 값만큼 나눈 나머지를 왼쪽 변수에 할당합니다.	a %= b	a = a % b

COUNT 8

익숙하지 않고, 혼란의 여지가 있을 경우에는 산술 연산자인 형태로 쓰셔도 됩니다. 실질적으로 연산의 결과값은 동일하기 때문에 상관 없습니다.

하지만 간결한 코드를 중요하게 여겨지는 순간이 온다면 그 때 대입 연산자로 활용하는 훈련을 조금씩 하면 됩니다. 물론 함께 일하는 팀의 컨벤션 등이 정해진다면 그것에 따르는 것이 좋습니다.

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. 비교 연산자

비교 연산자는 왼쪽 피연산자와 오른쪽 피연산자의 값을 비교하여 boolean 값(True, False)으로 반환해주는 역할을 합니다. 간단히 말하면 값을 비교하여 참과 거짓을 구분합니다.

오늘 잡은 물고기와 판매한 물고기의 수를 비교해 봅시다.

```
#[In]

오늘_잡은_물고기 = 10
오늘_판_물고기 = 5

print(오늘_잡은_물고기 > 오늘_판_물고기) # True
print(오늘_잡은_물고기 < 오늘_판_물고기) # False
print(오늘_잡은_물고기 >= 오늘_판_물고기) # True
print(오늘_잡은_물고기 <= 오늘_판_물고기) # False
print(오늘_잡은_물고기 == 오늘_판_물고기) # False
print(오늘_잡은_물고기 != 오늘_판_물고기) # True
```

Python ▾

비교 연산자는 **if** 조건문과 함께 가장 많이 사용됩니다. **if** 문은 뒤의 챕터에서 자세히 다룰 예정이니 이번 챕터에서는 어떻게 실행되는지만 간단히 알아봅시다.

if 문은 조건문이 참일 경우에만 실행문이 실행되며 거짓일 경우에는 실행되지 않습니다.

```
if 조건문 :
    실행문
```

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



앞의 예제에서 캣의 생선 가게에는 오늘 잡은 물고기를 당일 다 판매해야 하는 규칙이 있었죠?

```
오늘_잡은_물고기 = 10
오늘_판_물고기 = 5

if 오늘_잡은_물고기 == 오늘_판_물고기 :
    print('오늘 물고기 판매 끝!')

if 오늘_잡은_물고기 != 오늘_판_물고기 :
    print('아직 물고기를 다 팔지 않았습니다.')
```

Python ▾

이렇게 잡은 물고기의 수와 판매한 물고기의 수를 비교하여 값이 같지 않을 경우와 같을 경우에 따라 다른 문구를 출력할 수 있습니다.

위의 코드는 `if-else` 문을 통해 간결하게 표현할 수도 있습니다.

```
if 오늘_잡은_물고기 == 오늘_판_물고기 :
    print('오늘 물고기 판매 끝!')
else :
    print('아직 물고기를 다 팔지 않았습니다.')
```

Python ▾

`if` 문의 조건이 거짓일 경우에는 `if`문의 실행문이 실행되고 거짓을 경우에는 `else` 문의 실행문이 실행됩니다. 이 내용도 뒤에서 자세히 배울 내용이니 간단하게 알아보고 넘어가시길 바랍니다.

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



비교 연산자

Aa 기호	이름	설명	참(True)	거짓(False)
>	~보다 큰	왼쪽의 피연산자가 오른쪽의 피연산자보다 크면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	3 > 0	0 > 3
<	~보다 작은	왼쪽의 피연산자가 오른쪽의 피연산자보다 작으면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	3 < 0	3 > 0
>=	~보다 크거나 같은	왼쪽의 피연산자가 오른쪽의 피연산자보다 크거나 같으면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	3 >= 3 3 >= 0	0 >= 3
<=	~보다 작거나 같은	왼쪽의 피연산자가 오른쪽의 피연산자보다 작거나 같으면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	3 <= 3 3 <= 0	0 <= 3
==	같은	피연산자들이 같으면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	3 == 3	3 == 0
!=	다른	피연산자들이 다르면 참(True)을 반환하고 그렇지 않으면 거짓(False)을 반환합니다.	0 != 3	3 != 3

COUNT 6

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. 논리 연산자

논리 연산자는 boolean과 함께 쓰이며, 결과값으로 boolean 값(True, False)을 반환합니다. 논리 연산자에는 **and**, **or**, **not** 연산이 있습니다.

```
#[In]

a = True
b = False

print(a and a) # True
print(a and b) # False
print(a or a) # True
print(a or b) # True
print(not a) # False
print(not b) # True
```

Python ▾

and 연산은 피연산자 모두 참일 경우에만 True를 반환하며, **or 연산**은 피연산자 중 하나라도 참이면 True를 반환합니다. **not 연산**은 True일 때는 False를 False일 때는 True로 값을 반전시켜 반환합니다.

논리 연산자는 앞서 배웠던 비교 연산자와의 조합을 통해 조건을 더 까다롭고 명확하게 작성할 수 있습니다.

캐트의 생선 가게에는 또 다른 규칙이 있었죠? 바로 오늘 매출은 전날 매출의 2배를 목표로 한다는 규칙입니다. 앞의 예제에 규칙을 추가해 봅시다.

```
#[In]

오늘_잡은_물고기 = 10
오늘_판_물고기 = 10
전날_매출 = 10000
오늘_매출 = 20000

if (오늘_잡은_물고기 == 오늘_판_물고기) and (전날_매출*2 <= 오늘_매출):
    print('오늘 물고기 판매 끝!')
    print('매출 목표 달성!')
else :
    print('오늘의 목표를 달성하지 못했습니다.')
```

Python ▾

and 연산을 사용하여 모두 참일 경우에는 판매 종료와 매출 목표 달성을 알리고 둘 중 하나라도 거짓일 경우에는 "오늘의 목표를 달성하지 못했습니다."라는 문구가 출력됩니다.

or 연산을 사용할 경우에는 둘 중 하나만 조건을 만족하면 실행문이 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



```
#[In]

if (오늘_잡은_물고기 == 오늘_판_물고기) or (전날_매출*2 <= 오늘_매출):
    print('오늘 물고기 판매 끝!')
    print('매출 목표 달성!')
```

Python ▾

이번에는 not 연산을 사용해 봅시다. boolean 값을 변수에 저장하여 사용할 수도 있습니다.

```
#[In]

결과값 = 오늘_잡은_물고기 == 오늘_판_물고기

if 결과값 :
    print('오늘 물고기 판매 끝!')

if not 결과값 :
    print('아직 물고기를 다 팔지 않았습니다.')
```

Python ▾

논리 연산자

Aa 기호	이름	설명	참(True)	거짓(False)
and	그리고	양 쪽의 피연산자들 모두 참일 때만 참(True)을 반환하고 피연산자들 중 하나라도 거짓이면 거짓(False)을 반환합니다.	True and True	True and False False and True False and False
or	또는	양 쪽의 피연산자들 중 하나라도 참인 경우에 참(True)을 반환하고 피연산자들이 모두 거짓인 경우에만 거짓(False)을 반환합니다.	True or True True or False False or True	False or False
not	부정	오른쪽 피연산자의 논리 상태를 반전시킵니다.	not False	not True

COUNT 3

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



6. 비트 연산자

비트 연산자는 2진수로 이루어진 32비트의 0과 1로 이루어진 집합으로 표현을 하는 연산이며 진수 체계로 이루어진 숫자들(2진수, 8진수, 10진수, 12진수 등)로 계산하는 것이 아닙니다.

물론 값을 넣을 때는 숫자를 사용합니다. 조금 이해하기 어렵다면 2진수로 논리 연산을 하신다고 생각하시면 됩니다.

비트 연산자

Aa 기호	이름	설명
&	AND 연산	두 피연산자의 각 자리 비트의 값이 둘다 1인 경우 해당 자리에 1을 반환합니다. 하나라도 0인 경우에는 해당 자리에 0을 반환합니다.
	OR 연산	두 피연산자의 각 자리 비트의 값이 둘다 0인 경우 해당하는 자리에 0을 반환합니다. 하나라도 1인 경우에는 해당 자리에 1을 반환합니다.
^	XOR 연산	두 피연산자의 각 자리 비트의 값이 같을 경우 해당하는 자리에 0을 반환합니다. 두 피연산자의 각 자리 비트의 값이 다른 경우 해당하는 자리에 1을 반환합니다.
~	보수연산	피연산자의 각 자리 비트를 뒤집습니다.
<<	왼쪽 쉬프트 연산	오른쪽에서 0들을 이동시키면서 왼쪽 피연산자 이진수의 각 비트를 오른쪽 피연산자 비트만큼 왼쪽으로 이동시킨 값을 반환합니다.
>>	오른쪽 쉬프트 연산	사라지는 비트를 버리면서 왼쪽 피연산자 이진수의 각 비트를 오른쪽 피연산자 비트만큼 이동시킨 값을 반환합니다.

COUNT 6

💡 비트연산자는 초급 단계에서 사용할 일이 많이 없기 때문에 당장에 어려움을 느끼신다면 이 부분은 넘어가셔도 좋습니다.

```
#[In]

a = 2
b = 3

print(a & b) # 결과값 : 0
```

Python ▾

위에서 2와 3을 가지고 계산을 했습니다. 이것의 계산 과정을 살펴보겠습니다.

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



```

# 계산 과정

# a의 값 2를 2진수로 풀어줍니다.
0010

# b의 값 3을 2진수로 풀어줍니다.
0011

# 위아래로 붙여보겠습니다.
0010
0011

# & 연산의 경우 위와 아래 한쪽씩 맞춰 and 비교를 해봅시다.
# 이때 1 = True, 0 = False 입니다.
0010
0011
----
0010

# 결과값을 다시 10진수로 변환합니다.
2

```

Python ▾

이번에는 `|` 연산을 해보도록 하겠습니다.

```

#[In]

a = 7
b = 8

print(a | b) # 결과값 : 15

```

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



```

# 계산 과정

# a의 값 7을 2진수로 변환합니다.
0101

# b의 값 8을 2진수로 변환합니다.
1000

# 위아래로 붙여보겠습니다.
0101
1000

# 연산의 경우 위와 아래 한쪽씩 맞춰 or 비교를 해봅니다.
# 이때 1 = True, 0 = False 입니다.
0101
1000
----
1101

# 결과값을 다시 10진수로 변환해줍니다.
15
    
```

Python ▾

비트 연산자의 연산의 전개 과정은 다음과 같습니다.

1. 10진수를 2진수로 변환
2. 2진수에서 비트연산을 전개
3. 결과값을 다시 10진수로 변환

XOR 과 **보수연산** 또한 위의 전개 과정과 같기에 생략하도록 하겠습니다.

왼쪽 쉬프트 연산을 한 번 해보겠습니다. 이 때 주의할 점은 왼쪽 쉬프트 연산의 오른쪽 값은 2진수로 바꾸는 용도가 아닙니다. 그 만큼 비트 이동을 하겠다는 의미입니다.

```

#[In]

a = 7
b = 2

print(a << b) # 결과값 :28
    
```

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



```

# 계산 과정

# a의 값 3을 2진수로 바꿉니다.
0111

# b 만큼 이동할 것이기 때문에 b는 바꾸지 않음

# a의 값들(집합)을 왼쪽으로 b만큼 이동시키고 뒤에 0을 붙임
011100

# 결과 값을 10진수로 다시 변환합니다.
28

```

Python ▾

이번에는 오른쪽 쉬프트 연산을 해보도록 하겠습니다.

```

#[In]

a = 7
b = 2

print(a >> b) # 결과값 : 1

```

Python ▾

```

# 계산 과정

# a의 값 7을 2진수로 바꿉니다.
0111

# b만큼 이동할 것이기 때문에 b는 바꾸지 않음

# a의 값들(집합)을 오른쪽으로 b만큼 이동시키고 넘어간 것을 잘라냄
0001(11) # ()는 넘어간 것을 의미합니다. 이것들은 잘라내어 버립니다.

0001

# 결과 값을 10진수로 다시 변환합니다.
1

```

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



7. 멤버 연산자

멤버 연산자는 특정 항목이 배열에 들어있는지 확인하는 연산입니다. 결과값이 참이면 True, 거짓이면 False를 반환합니다.

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

```
#[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.

```
# 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 연산자입니다. 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 주소라고 합니다. 저희들이 사용하는 주소와 비슷한 개념이라고 보시면 됩니다. 00시 00동 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 다르다면 True 그렇지 않으면 False

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

```
#[In]

위니브월드 = "licat"
바울랩 = "licat"

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

Python ▾

같다면 동일인물이라는 것을 알 수 있습니다.

즉, 이 두 회사에 다니고 있는 `licat` 이라는 이름을 가진 사람은 00시 00동 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



```

#[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 ▾

```

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

```
#[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





갯의 해골섬 출항!



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?

```

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

Python ▾

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.

```

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

Python ▾

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.

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

Python ▾

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.

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

Python ▾

모두 완료하였다면 아래 훈장을 추가하세요!

```
훈장.append('해골섬 낚시꾼')
```

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





3편 효율적으로 생선 잡기

1. 캣의 고민
 - 1.1 들여쓰기
 - 1.2 함수의 사용
 - 1.3 print VS return
2. 함수를 사용한 캣의 계산
 - 2.1 상수(Constant)
3. 전역변수 global
4. function 응용
 - 4.1 함수 안에 함수 만들기
5. 연산자 우선순위

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 귀뜸을 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!"

```
#[In]
```

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

Python ▾

- 라이캣의 현재 상태창!

```
#[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.

```
#[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 범위를 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 :

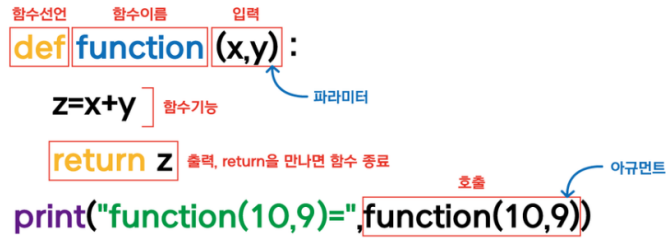
DATE

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.

```
#[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

```
#[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



2. 함수를 사용한 컷의 계산

```
#[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.

```
#[Out]

5000 원입니다.
```

Python ▾

등급 가격은 정해져 있는데 매번 입력하는 것은 귀찮다냥!

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

```
#[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.

```
#[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** 수입니다. C언어에서는 define이라는 문장을 사용하여 상수를 정의하지만, Python에서는 이러한 상수를 선언하는 문법이 없습니다. 다만 **일반적으로 이를 전부를 대문자로 선언하면 상수**라고 이해합니다.

```
#[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.

```
#[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.

```
#[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.

```
#[In]
```

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

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[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.

```
#[In]

A등급 = 0

def 낚시():
    global A등급
    A등급 += 1

낚시()
낚시()
낚시()

print('잡은 A등급 :', A등급, '마리')
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[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.

```
#[In]

# 낚시, 그물, 통발 모두 사용하여 잡은 생선의 총 합계 구하기

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.

```
#[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.

```
#[In]

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

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.

```
#[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.

```
#[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.

```
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

Aa 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





4편 생선 회사를 운영하라

1. 낫의 생선회사
 - 1.1 낫의 회사 운영
 - 1.2 if 문의 기본 구조
2. if, else 문
3. if, elif, else

1. 캣의 생선 회사

'캣'의 생선가게는 어느새 마을에서 가장 유명한 생선가게가 되었다!



캣의 생선 가게는 어느새 마을에서 가장 유명한 생선 가게가 되었습니다. 장사가 잘 되어 기쁜 캣은 한편으로는 걱정이 되었습니다. 캣의 작은 가게는 불어난 손님을 다 수용하기에는 역부족이었기 때문입니다.

"바쁘다 바빠! 이젠 작은 생선가게에서 벗어날 때가 된 것 같다냥!"

캣은 작은 생선가게를 큰 생선회사로 업그레이드 시키기로 결심하였습니다.

"생선회사로 업그레이드! 체계적인 관리와 시스템으로 많은 돈을 벌어보겠다냥! 그 전에, 직원부터 뽑자냥!"

- 라이캣의 상태창!

```
#[In]

이름 = '캣'
설명 = '위니브 월드 외각에 살고 있는 생선가게 주인 캣(cat)'
나이 = 17
오늘_잡은_물고기 = '10000'
키 = '45.7cm'
몸무게 = 1.4
육식 = True
초식 = True
돈 = 100000
훈장 = ['백상아리를 잡은 고양이', '성실한 납세자', '해골섬 낚시꾼']
기술 = ['고기잡이', '고기팔기', '낚시_Lv3', '통발_Lv3', '큰그물_Lv3']
```

Python ▾

- 라이캣의 기술 Update!

```
#[In]

기술[2] = 기술[2][:-1] + '3'
기술[3] = 기술[3][:-1] + '3'
기술[4] = 기술[4][:-1] + '3'
```

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



1.1 캣의 회사 운영

직원을 채용하기로 한 캣은 직원을 뽑을 때 성실성과 프로그래밍 능력을 보기로 합니다. 성실성이 80점 이상이고 프로그래밍 능력이 70점 이상일 경우에 합격입니다.

```
#[In]

#캣의 직원 채용

성실성 = 85
프로그래밍능력 = 70

if 성실성 >= 80 and 프로그래밍능력 >= 70:
    print('합격입니다.')
```

Python ▾

위 코드를 실행하면 아래와 같이 출력됩니다.

```
#[Out]

합격입니다.
```

Python ▾

이처럼 조건을 판단하여 참일 경우에 해당 조건에 맞는 것을 실행하는 것을 if문이라고 합니다.

위 예제에서 조건문은 `성실성 >= 80 and 프로그래밍능력 >= 70` 입니다. 성실성은 85이고 프로그래밍 능력은 70이므로 조건에 만족하여 `print('합격입니다.')` 가 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



생선 회사의 매출은 모두 손님 덕분! 보답하겠다냥!

직원 채용으로 차츰 안정되고 매출도 꾸준히 늘고있는 캣의 생선회사. 캣은 손님들을 향한 고마움을 '할인 제도'를 통해 보답하고자합니다.

"만원이상 구입하면 천원을 할인한다냥!"

```
#[In]

#생선회사의 할인제도
# A등급:1000원, B등급:500원, C등급:100원

def 계산(a, b, c):
    가격 = {'A등급':1000, 'B등급':500, 'C등급':100}
    합계 = a*가격['A등급'] + b*가격['B등급'] + c*가격['C등급']
    return 합계

총합 = 계산(10, 2, 3)

if 총합 >= 10000:
    print('할인을 해주겠다냥!')
    print(f'총 {int(총합 * 0.9)}노드를 내면 된다냥!') #10% 할인
```

Python ▾

위 코드를 실행하면 아래와 같이 출력됩니다.

```
#[Out]

할인을 해주겠다냥!
총 10170노드를 내면 된다냥!
```

Python ▾

위 예제에서 함수 `계산(a, b, c)` 를 통해 총합을 구합니다. 조건문은 `총합 >= 10000` 으로 총합이 13000으로 10000 이상이므로 출력문이 실행됩니다.

이 취업난 시대에 한명의 청년이라도 더 고용하는 라이캣에게 훈장을 수여하도록 하겠습니다.

```
#[In]

훈장.append('청년 고용 착한 기업')
훈장.append('회사를 설립한 자')
```

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



1.2 if 문의 기본 구조

배운 내용을 if 문의 기본 구조를 통해 개념을 정리하고 넘어갑시다.

```
#if 문의 기본구조
if 조건문 :
    수행할 문장 1
    수행할 문장 2
```

Python ▾

조건문이 참이면 **수행할 문장 1** 과 **수행할 문장 2** 를 수행합니다.

```
if 조건문 :
    수행할 문장 1
    수행할 문장 2
```

Python ▾

위의 if문에서는 조건문이 참일 경우에 **수행할 문장 1** 을 수행하고 **수행할 문장 2** 는 **if문 범위 밖에** 있기 때문에 조건문에 상관없이 수행됩니다.

```
if True:
    print('True')
```

Python ▾

또한 위 예제 코드처럼 조건문에 직접 boolean 값(True, False)을 넣을 수도 있습니다.

Project name :

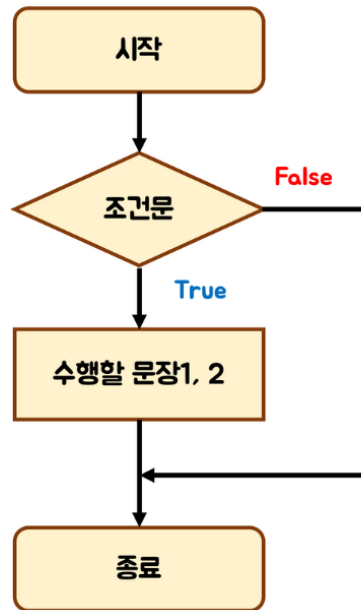
DATE

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



- if 문의 순서도(Flow Chart)



2. if, else 문

else 문은 if 문의 조건이 거짓일 경우에 실행됩니다. 아래 if, else 문의 기본 구조를 통해 알아보시다.

```

#if, else문의 기본구조
if 조건문:
    수행할 문장 1
    수행할 문장 2
else:
    수행할 문장 3
    수행할 문장 4
    
```

Python ▾

if 문의 조건문을 확인해서 만약 참이면 수행할 문장 1 과 수행할 문장 2 를 수행하고, 조건이 거짓이면 else 문이 실행되어 수행할 문장 3 과 수행할 문장 4 가 수행됩니다.

Project name :

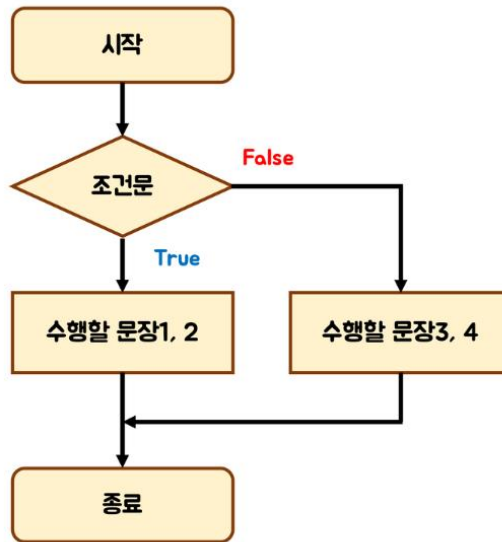
DATE

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



- if, else 문의 순서도(Flow Chart)



```

#[In]

#컷트의 직원 채용
성실성 = int(input('성실성을 입력하세요 :'))
프로그래밍능력 = int(input('프로그래밍능력을 입력하세요 :'))

if 성실성 >= 80 and 프로그래밍능력 >= 70:
    print('합격입니다.')
    if 성실성 >= 90:
        print('보너스를 드립니다.')
else:
    print('불합격입니다.')
    
```

Python ▾

위 코드를 실행하면 아래와 같이 출력됩니다.

```

#[Out]

성실성을 입력하세요 :80
프로그래밍능력을 입력하세요 :80
합격입니다.
    
```

Python ▾

위 예제에서는 성실성과 프로그래밍 능력을 `input()` 함수를 통해 입력받습니다.

if 문의 조건문을 만족할 경우 `합격입니다.` 가 출력됩니다. 조건문을 만족하지 못할 경우에는 else 문으로 넘어가 `불합격입니다.` 가 출력됩니다.

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 성실성과 프로그래밍 능력을 `input()` 함수를 통해 입력받습니다.

if 문의 조건문을 만족할 경우 `합격입니다.` 가 출력됩니다. 조건문을 만족하지 못할 경우에는 else 문으로 넘어가 `불합격입니다.` 가 출력됩니다.

💡 `input()` 은 Python에서 입력을 받을 때 사용하는 함수이며, 입력받은 값을 `int()` 를 통해 정수형으로 변환할 수 있습니다.

이번에는 총합을 입력받고 else문을 추가하여 조건이 거짓일 경우에는 할인을 하지 않은 금액을 출력해 봅시다.

```
#[In]

#생선회사의 할인제도
총합 = int(input('가격을 입력하세요 :'))

if 총합 >= 10000:
    print('할인을 해주겠다냥!')
    print(f'총 {int(총합 * 0.9)}노드를 내면 된다냥!') #10% 할인
else:
    print(f'총 {총합}노드를 내면 된다냥!')
```

Python ▾

위 코드를 실행하면 아래와 같이 출력됩니다.

```
#[Out]

# 1
가격을 입력하세요 :20000
할인해드립니다!
총 18000 원입니다.

# 2
가격을 입력하세요 :9000
총 9000 원입니다.
```

Python ▾

위 예제에서는 총합을 입력받고 if 문에서 조건을 검사합니다. 총합이 10000 이상일 경우에는 if 문 다음 문장이 수행되고 거짓일 경우에는 else 문이 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



3. if, elif, else

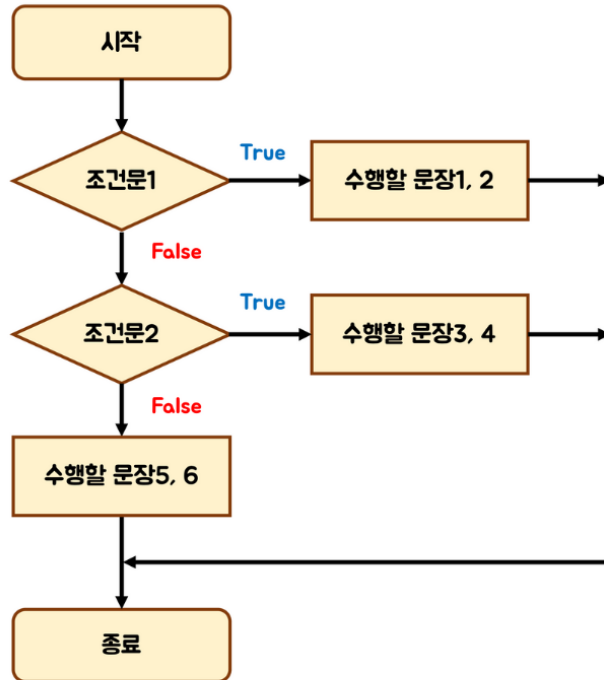
```
#if,elif,else문의 기본구조

if 조건문1:
    수행할 문장 1
    수행할 문장 2
elif 조건문2:
    수행할 문장 3
    수행할 문장 4
else:
    수행할 문장 5
    수행할 문장 6
```

Python ▾

if 문에서 조건문을 확인해서 만약 참이면 **수행할 문장 1** 과 **수행할 문장 2** 를 수행합니다. 그렇지 않고 거짓일 경우에는 elif 문의 조건문을 확인합니다. elif 문에서도 마찬가지로 조건문이 참일 경우에는 **수행할 문장 3** 과 **수행할 문장 4** 를 수행하고 거짓이면 else 문이 실행됩니다.

- if, elif, else문의 순서도(Flow Chart)



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 가격과 손님에게 받은 돈을 입력받아 작성해 봅시다.

```
#[In]

# 거스름돈
가격 = int(input('가격을 입력하세요 :'))
받은돈 = int(input('받은돈을 입력하세요 :'))

if 가격 < 받은돈:
    print('여기 거스름돈', 받은돈-가격, '원입니다.')
elif 가격 > 받은돈:
    print(가격-받은돈, '원이 부족합니다.')
else:
    print('감사합니다.')
```

Python ▾

위 코드를 실행하면 아래와 같이 출력됩니다.

```
#[Out]

# 1
가격을 입력하세요 :9000
받은돈을 입력하세요 :10000
여기 거스름돈 1000 원입니다.

# 2
가격을 입력하세요 :9000
받은돈을 입력하세요 :8000
1000 원이 부족합니다.

# 3
가격을 입력하세요 :9000
받은돈을 입력하세요 :9000
감사합니다.
```

Python ▾

위 예제에서는 **가격** 과 **받은돈** 을 입력받아 둘의 값을 비교합니다. 가격보다 받은돈이 클 경우에는 **받은돈-가격** 의 연산 결과인 거스름돈을 출력하고, 작을 경우에는 돈이 부족하다는 문구를 출력합니다.

if 문과 elif 문의 조건이 모두 거짓일 경우 즉, 가격과 받은돈의 값이 같을 경우에는 "감사합니다."라고 출력합니다.

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



위에서 배운 내용을 가지고 할인 내역을 좀 더 세분화 해봅시다.

```
#[In]

# A등급:1000원, B등급:500원, C등급:100원

def 계산(a, b, c):
    가격 = {'A등급':1000, 'B등급':500, 'C등급':100}
    합계 = a*가격['A등급'] + b*가격['B등급'] + c*가격['C등급']
    return 합계

총합 = 계산(20, 2, 3)
print(총합)

if 총합 >= 20000:
    print('20% 할인을 해주겠다냥!')
    print(f'총 {int(총합 * 0.8)}노드를 내면 된다냥!') #20% 할인
elif 총합 >= 10000:
    print('10% 할인을 해주겠다냥!')
    print(f'총 {int(총합 * 0.9)}노드를 내면 된다냥!') #10% 할인
elif 총합 >= 5000:
    print('5% 할인을 해주겠다냥!')
    print(f'총 {int(총합 * 0.95)}노드를 내면 된다냥!') #5% 할인
else:
    print(f'총 {총합}노드를 내면 된다냥!')
```

Python ▾

위 코드의 실행 결과는 아래와 같습니다.

```
#[Out]

21300
20% 할인을 해주겠다냥!
총 17040노드를 내면 된다냥!
```

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





갯의 고객관리



갯은 효율적으로 고객들을 관리하기 위해 고객들을 회원으로 등록하고 구매 금액의 10%를 적립해주는 "적립금 제도"를 시작하였습니다.

단, 적립금은 회원인 경우에만 적립할 수 있으며 10,000노드 이상이 모이면 생선 구매 시 현금처럼 사용할 수 있습니다.

1. 회원으로 등록하겠습니까?

손님 'A갯'과 'D갯'은 갯의 생선 회사에 회원인지 확인하고 싶습니다. 만약 회원이 아닐 경우에는 회원으로 등록하고 싶다고 합니다.

아래 코드를 작성하여 'A갯'과 'D갯'이 회원인지 확인하고 회원으로 등록해 주세요.

```
#[In]

회원명 = input('회원명 입력하세요 : ')
회원 = ['A갯', 'B갯', 'C갯']

def 회원등록():
    '정답을 입력해주세요.'

if '정답을 입력해주세요.':
    '정답을 입력해주세요.'
else:
    회원등록()
```

Python ▾

```
#[Out]
```

```
회원명 입력하세요 : A켓  
이미 등록된 회원입니다.
```

```
---
```

```
회원명 입력하세요 : D켓  
['A켓', 'B켓', 'C켓', 'D켓']  
회원으로 등록되었습니다.
```

Python ▾

2. 회원에게 적립금을 지급하라!

손님 '애옹'이와 '냐옹'이는 생선을 각각 15,000노드, 5,000노드씩 구입하였습니다. 이때 '애옹'이와 '냐옹'이가 회원인 경우를 체크합니다. 회원인 경우 적립받게 되는 금액을 출력하고, 회원이 아닌 경우에는 아직 회원이 아님을 출력해주세요.

```
#[In]
```

```
구매가격 = input('가격을 입력하세요 :')  
회원명 = input('회원명을 입력하세요 :')  
  
회원 = ['씨-켓', '자바켓', '파이켓', '썬켓', '애옹']
```

Python ▾

```
#[Out]
```

```
회원명 입력하세요 : 애옹  
회원입니다.  
15000노드 중 1500노드가 적립됩니다.
```

```
---
```

```
회원명 입력하세요 : 냐옹  
회원이 아닙니다. 회원가입을 해주시기 바랍니다.
```

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. 적립금을 사용하고 싶어요!

손님 '파이캣'과 '썬캣'은 생선을 각각 10,000노드씩 구매하였습니다. 두 손님은 적립금을 사용할 수 있는지 알고 싶습니다.

사용할 수 있다면 적립금 5,000노드를 사용합니다. 이때 구매 가격과 남은 적립금은 얼마인지 출력하고 남은 적립금은 Dictionary에 반영해 주세요.

그렇지 않을 경우에는 현재 적립금은 얼마인지 출력해주세요.

```
#[In]

구매가격 = input('가격을 입력하세요 :')
회원명 = input('회원명을 입력하세요 :')

#회원 = {회원명:적립금}
회원 = {'씨-캣': 5000, '자바캣': 3500, '파이캣': 15000, '썬캣': 7000}
```

Python ▾

```
#[Out]

In 가격을 입력하세요 : 10000
In 회원명을 입력하세요 : 파이캣
Out 적립금을 사용할 수 있습니다. 현재 적립금액은 15000노드 입니다.
In 사용할 적립금 노드를 입력하세요 : 5000
Out 5000노드를 사용하였습니다. 남은 적립금은 10000노드입니다. 결제금액은 5000원 입니다.
```

```
가격을 입력하세요 : 10000
회원명을 입력하세요 : 썬캣
현재 적립금이 7000노드이므로 아직 적립금을 사용할 수 없습니다. 결제금액은 10000노드이고, 적립 포인트는 1000노드, 합산 포인트는 8000 노드입니다.
```

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



4. 적립금 이벤트를 진행합니다!(심화, google에서 Factory 함수라고 검색해보세요.)

캣은 매달 월과 같은 요일에 적립금 이벤트를 진행하기로 했습니다.

예를들어, 2월 2일에는 적립금의 2배를, 3월 3일에는 적립금의 3배를 적립해줍니다.

만약 손님이 2월 2일에 5000원의 생선을 구매하고, 3월 3일에 15000원을 구매 했다면 각각 얼마의 적립금을 받을 수 있을까요? 중첩 함수를 사용하여 풀어보세요!

```
def 배수(n):
    def 적립(value):
        '정답을 입력하세요.'
    return 적립

Feb = 배수(2)
Mar = 배수(3)

print('2월 적립금 이벤트 :', '정답을 입력하세요.')
print('3월 적립금 이벤트 :', '정답을 입력하세요.')
```

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





5편 라이캣의 EXIT

1. 캣의 로봇(for)
 - 1.1 for문의 기본 구조
 - 1.2 for, else
 - 1.3 지능형 리스트(list comprehension)의 for
 - 1.4 다중인자 리스트 순회
 - 1.5 enumerate
 - 1.6 계산하는 로봇
2. 캣의 로봇(while)
 - 2.1 무한반복 while, break
 - 2.2 while, else
3. break
 - 3.1 break 문
4. continue, pass
5. else
6. 중첩 반복문

1. 캣의 로봇(for)



캣의 생선회사는 어느덧 유니브 월드 전체에서 가장 빠르게 성장하는 생선회사가 되었습니다. 대부분의 스타트업이 그렇듯이 성장세에 일만하다 보니 높은 연봉에도 직원들의 불만이 늘어가고 있었습니다.

"자율성과 창의적 생각이 보장되고, 개인이 성장하면서, 각자 하는 일에 대한 명료한 목적과 동기부여를 줄 수는 없을까냥?"

캣은 어느새 대표다운 생각을 하고 있었어요. 매주 아침 하는 회의에서 캣은 말했습니다.

"생산성을 극대화 하면서도, 직원들의 휴식시간을 늘려줄 수 있는 로봇을 만들어보는 것은 어떨까냥?"

하지만 이미 불만이 가득차 있는 직원들은 냉담한 반응이었습니다.

"누가 만드냐!? 대표가 만드냐!?"

캣은 더이상의 회의가 무의미하다는 생각에 회의를 종료했습니다. 그리고, 정말 혼자 만들기 시작했어요.

"밤을 새서라도 다음주까지 만들겠다냥!"

- 라이캣의 상태창!

```
#[In]

이름 = '캣'
설명 = '위니브 월드에서 가장 급성장하는 생선회사 대표 캣'
나이 = 19
오늘_잡은_물고기 = '100000'
직원수 = 4
키 = '45.9cm'
몸무게 = 1.6
잡식 = True
돈 = 1100000
훈장 = ['백상아리를 잡은 고양이', '성실한 납세자', '해골섬 낚시꾼', '청년 고용 착한 기업', '회사를 설립한 자']
기술 = ['고기잡이', '고기팔기', '낚시_Lv5', '통발_Lv5', '큰그물_Lv5']
```

Python ▾

캣은 가용한 모든 자원, 알고 있는 모든 지식을 투입하기 시작했어요. 아래 코드를 봅시다.

```
#[In]

for x in (1,2,3,4,5):
    print(f'고등어 포장 {x}번째 입니다.')
```

Python ▾

우선 직원들이 가장 힘들어하는 고등어 포장 로봇을 만들기 시작했어요. `for` 라는 강력한 프로그래밍 문법을 사용하였습니다.

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

고등어 포장 1번째 입니다.
고등어 포장 2번째 입니다.
고등어 포장 3번째 입니다.
고등어 포장 4번째 입니다.
고등어 포장 5번째 입니다.
```

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 코드처럼 정해진 횟수를 반복하는 것을 for문이라고 합니다. 코드를 조금 더 살펴보죠. `x` 라는 변수를 선언하여 튜플 `(1,2,3,4,5)` 의 길이만큼 for문 아래에 있는 문장을 반복하게 됩니다.

또, 변수 `x` 는 튜플 요소의 값을 하나씩 가지게 되죠. 조금 어렵죠? 더 자세한 설명은 다음 챕터에서 하도록 하겠습니다.

1.1 for문의 기본 구조

위에서 배운 for문의 기본 구조를 더 자세하게 정리하고 넘어가도록 하겠습니다. for문은 순서열을 순회하며 순서열의 끝에 도달하면 반복을 멈추게 됩니다. 또한 객체를 처음부터 끝까지 하나씩 **추출하며 순회**하기 때문에 그 사용법이 쉬워 가장 많이 사용되는 반복문입니다.

다음 챕터에서 배울 while은 비교할 변수를 먼저 선언 해주어야 하기 때문에, 비교적 for를 더 많이 사용합니다. 하지만 각자의 용법이 있어, 어느 문법이 좋다고는 할 수 없습니다!

```
#[In]

for x in (1,2,3):
    print(f'{x}번째 로봇을 똑똑똑')
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

1번째 로봇을 똑똑똑
2번째 로봇을 똑똑똑
3번째 로봇을 똑똑똑
```

Python ▾

위의 코드를 실행하면 변수 `x`에는 튜플(1, 2, 3)의 요소가 순서대로 출력됩니다. 위 예제에서 튜플의 길이는 3이므로 for문은 반복을 3번 수행합니다. 여기서 `x`는 다른 언어처럼 초기화 하지 않아도 작동합니다.

```
#for 문의 구조
for (변수명) in (순회 가능한 객체) :
    수행할 문장1
    수행할 문장2
```

Python ▾

for 문의 범위로 사용되는 것은 시퀀스 자료형 자료 또는 반복 가능한 자료형이어야 합니다.

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



- 반복문 순서도(Flow Chart) - for 문



1. String(문자열)을 범위로 지정한 예시

```

#[In]

왕국 = '위니브 월드'
for a in 왕국 :
    print(a)
    
```

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



```
#[Out]
```

```
위  
니  
프  
델  
크
```

Python ▾

2. List(리스트)를 범위로 지정한 예시

```
#[In]
```

```
기술 = ['고기잡기', '고기팔기', '낙시_Lv1', '통발_Lv1', '큰그물_Lv1']  
for a in 기술:  
    print(a)
```

Python ▾

```
#[Out]
```

```
고기잡기  
고기팔기  
낙시_Lv1  
통발_Lv1  
큰그물_Lv1
```

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. Dictionary(사전)을 범위로 지정한 예시

```
#[In]

켓의_상태창 = {
    '이름': '켓',
    '설명': '위니브 월드에서 가장 급성장하는 생선회사 대표 켓',
    '나이': '19',
    '키': '45.9cm',
    '몸무게': 1.6
}

for a in 켓의_상태창:
    print(a)
```

Python ▾

#[Out]

```
이름
설명
나이
키
몸무게
```

Python ▾

위의 예제에서 Dictionary(사전)형 자료의 경우에는 key(키)만을 가져오게 됩니다.

key(키)에 해당하는 value(값) 또한 가져오고 싶다면 아래와 같이 튜플 언패킹을 사용할 수 있습니다.

```
#[In]

for key,value in 켓의_상태창.items():
    print("{0} : {1}".format(key, value))
```

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]


이름 : 캣
설명 : 위니브 월드에서 가장 급성장하는 생선회사 대표 캣
나이 : 19
키 : 45.9cm
몸무게 : 1.6
```

Python ▾

for문에서 가장 많이 사용되는 range에 대해 알아보시다. range는 특정 범위를 생성하기 위해 사용할 수 있습니다.

 range는 2.x에서는 선언하는 즉시 list가 되었으나 3.x으로 넘어오면서 range는 list로 변환해 주어야 list가 됩니다. 이는 2.x에서 썼었던 xrange와 같은 함수입니다.

range 함수는 연속하는 수열을 만듭니다. `range(시작_값, 종료_값, 연속하는_두_수의_차)` 형식으로 선언되며 아래 예제를 보며 자세히 설명해 드리도록 하겠습니다.

 `range(start, stop, step)`으로 주요 표현합니다. 어디서 많이 보았던 형식이죠? 바로 슬라이싱에서 보았던 형식입니다.

시작 값 : 0, 종료 값 : 5, 연속하는 두 수의 차 : 1

```
#[In]

for a in range(0, 5, 1):
    print(a)
```

Python ▾

```
#[Out]
```

```
0
1
2
3
4
```

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



시작값 0과 멈춤값 5 사이의 연속하는 두 수의 차 1을 가지고 있는 요소들이 생성되었습니다. 생성된 range의 마지막 요소가 멈춤값보다 한 단계 작다는 사실에 주의하세요.

시작 값 : 0, 종료 값 : 10, 연속하는 두 수의 차 : 2

#[In]

```
for a in range(0, 10, 2):
    print(a)
```

Python ▾

#[Out]

```
0
2
4
6
8
```

Python ▾

시작 값 0과 종료 값 10 사이의 연속하는 두 수의 차 2를 가지고 있는 요소들이 생성되었습니다. 생성된 range의 마지막 요소가 종료 값보다 한 단계 작다는 사실에 주의하세요. range()함수의 마지막 매개변수인 연속하는 두 수의 차는 생략할 수 있습니다. 생략할 시 연속하는 두수의 차는 1입니다.

시작 값 : 0, 종료 값 : 5, 연속하는 두수의 차 : 생략

#[In]

```
for a in range(0, 5):
    print(a)
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

#[Out]

```
0
1
2
3
4
```

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



마지막 매개변수인 연속하는 두 수의 차를 생략하고 range() 함수를 호출하였습니다. 이 경우 range() 함수의 마지막 매개변수인 연속하는 두 수의 차는 1이 됩니다.

시작 값 : 생략, 종료 값 : 5, 연속하는 두 수의 차 : 생략

```
#[In]

for a in range(5):
    print(a)
```

Python ▾

```
#[Out]
```

```
0
1
2
3
4
```

Python ▾

또한 range() 함수는 종료 값만을 입력하여 호출할 수 있습니다. 종료 값만 입력했을 시에는 시작값은 0이 되며 두 수의 차는 1이 됩니다.

시작 값 : 10, 종료 값 : 5, 연속하는 두 수의 차 : -1

```
#[In]

for a in range(10, 5, -1):
    print(a)
```

Python ▾

```
#[Out]
```

```
10
9
8
7
6
```

Python ▾

연속하는 두 수의 차를 마이너스로 둘 수도 있습니다. 이 경우 시작 값과 종료 값 보다 크게 주셔야 한다는 점도 기억해주세요.

좀 더 알아보을까요? 알은물에서 할 내용은 아니니 가볍게 읽고 넘어가주세요. for 문에 사용될 수 있는 시퀀스형 자료는 문자열, 리스트, 튜플이 있습니다. 이 자료형은 메서드로 `__iter__`와 `__next__`를 가지고 있으며 이 두개의 메서드로 순회가 가능하게 합니다.

이는 아래와 같이 `iter()`함수와 `next()`함수로도 실행을 할 수 있습니다.

```
#[In]

listx= [100,200,300,400]
strx= 'abcd'
listxIter = iter(listx)
strxIter= iter(strx)
print(next(listxIter),next(listxIter),next(listxIter),next(listxIter))
print(next(strxIter),next(strxIter),next(strxIter),next(strxIter))
```

Python ▾

위 결과값은 아래와 같습니다.

```
#[Out]

100 200 300 400
a b c d
```

Python ▾

1.2 for, else

```
#[In]

for i in range(100):
    print(f'{i} 물고기를 잡았습니다.')
    if i == 5:
        print('만선입니다. 물고기를 다 잡았습니다.')
        break
else:
    print('아직 여유가 좀 있지만, 물고기가 더 없는 것 같으니 이정도로 만족하고 돌아갑시다.')
print('수고하셨습니다.')
```

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]

0마리의 물고기를 잡았습니다.
1마리의 물고기를 잡았습니다.
2마리의 물고기를 잡았습니다.
3마리의 물고기를 잡았습니다.
4마리의 물고기를 잡았습니다.
5마리의 물고기를 잡았습니다.
만선입니다. 물고기를 다 잡았습니다.
```

Python ▾

```
#[In]

for i in range(5):
    print(f'{i} 물고기를 잡았습니다.')
    if i == 5:
        print('만선입니다. 물고기를 다 잡았습니다.')
        break
else:
    print('아직 여유가 좀 있지만, 물고기가 더 없는 것 같으니 이정도로 만족하고 돌아갑시다.')
    print('수고하셨습니다.')
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다. 위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

0마리의 물고기를 잡았습니다.
1마리의 물고기를 잡았습니다.
2마리의 물고기를 잡았습니다.
3마리의 물고기를 잡았습니다.
4마리의 물고기를 잡았습니다.
아직 여유가 좀 있지만, 물고기가 더 없는 것 같으니 이정도로 만족하고 돌아갑시다.
수고하셨습니다.
```

Python ▾

if문 뿐만 아니라 for에서도 else를 사용할 수 있습니다. else는 루프가 정상 종료되었을 때, 처음부터 자료형이 비어있었을 때 실행됩니다. break문을 만나면 else 문을 실행하지 않고 빠져나옵니다.

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 지능형 리스트(list comprehension)

얇은물에서는 가볍게 이런 것이 있다는 것만 알고 넘어가도록 하겠습니다. 가장 많이 사용되는 list 생성 기법입니다. 'list comprehension'은 한국어로 표현할 때 리스트 표현식, 지능형 리스트로 번역이 되곤 합니다.

```
#[In]

#1
x= [i for i in range(1, 10)]
print(x)

#2
y=[ '{X}X{Y}={Z}'.format(i, j, i*j) for i in range(2, 10) for j in range(1, 10)]
print(y)

#3
def sumthingFunction(i):
    if i % 100 ==0:
        return i
    else:
        return 0
기존리스트= [100,200,300,101,202,303]
새로운리스트= [sumthingFunction(i) for i in 기존리스트]
print(sum(새로운리스트))
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

#1
[1, 2, 3, 4, 5, 6, 7, 8, 9]

#2
구구단 리스트 출력

#3
600
```

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



1.4 다중인자 리스트 순회

```
#[In]

#다중 리스트 for문
skill = [
    ('고기잡이', 100),
    ('고기팔기', 120),
    ('낚시', 5),
    ('통발', 5),
    ('큰그물', 5)
]

for i,j in skill:
    print(i,j)
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

고기잡이 100
고기팔기 120
낚시 5
통발 5
큰그물 5
```

Python ▾

```
#[In]

#하나만 리스트여도 쌍이면 상관없이 잘 돌아감
skill = [
    ('고기잡이', 100, 'SS'),
    ('고기팔기', 120, 'SSS'),
    ('낚시', 5, 'C'),
    ('통발', 5, 'C'),
    ('큰그물', 5, 'C')
]

for skillName, skillLevel, skillGrade in skill:
    print(skillName, skillLevel, skillGrade)
```

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]

고기잡이 100 SS
고기팔기 120 SSS
낚시 5 C
통발 5 C
큰그물 5 C
```

Python ▾

1.5 enumerate

enumerate는 순서를 매길 때 사용합니다. 각각의 스킬들을 습득한 순서대로 출력하고 싶어요. 그럼 어떻게 출력을 해줄 수 있을까요? enumerate는 별도의 변수를 선언하지 않고 이것이 가능하게 해줍니다.

```
#[In]

# enumerate

skill = [
    ('고기잡이', 100, 'SS'),
    ('고기팔기', 120, 'SSS'),
    ('낚시', 5, 'C'),
    ('통발', 5, 'C'),
    ('큰그물', 5, 'C')
]

for i, (skillName, skillLevel, skillGrade) in enumerate(skill):
    print(i, skillName, skillLevel, skillGrade)

for i, j in enumerate(skill, 100):
    print(i, j)
```

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]

0 고기잡이 100 SS
1 고기팔기 120 SSS
2 낚시 5 C
3 통발 5 C
4 큰그물 5 C
100 ('고기잡이', 100, 'SS')
101 ('고기팔기', 120, 'SSS')
102 ('낚시', 5, 'C')
103 ('통발', 5, 'C')
104 ('큰그물', 5, 'C')
```

Python ▾

1.6 계산하는 로봇

캣은 로봇을 이용해서 '고등어 포장' 보다 조금 더 어려운 '카운터 보기'를 맡기로 했어요. 손님께 생선 몇 개를 구매하실 건지 여쭙보고 그 갯수에 맞는 금액을 출력할 수 있는 로봇을 만들었어요.

```
#[In]

for i in range(5):
    num = int(input("고등어 몇 개를 구매하실 건가요?"))
    result = num * 5000
    print(result, "원 입니다.")

print("로봇이 종료되었습니다. 빼빅.")
```

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]

고등어 몇 개를 구매하실 건가요? 5
25000원 입니다.
고등어 몇 개를 구매하실 건가요? 3
15000원 입니다.
고등어 몇 개를 구매하실 건가요? 19
95000원 입니다.
고등어 몇 개를 구매하실 건가요? 1
5000원 입니다.
고등어 몇 개를 구매하실 건가요? 7
35000원 입니다.
로봇이 종료되었습니다. 배빅.
```

Python ▾

고등어를 몇 개 구매할 건지 `input()` 명령어를 통해 손님께 여쭙보고, 그 갯수에 가격을 곱해서 출력을 해주도록 만들었습니다.

2. 캣의 로봇(while)

금방 배웠던 `for`문을 이용해서 로봇을 만들면 정해진 숫자에서 종료되기 때문에, 생선이 더 있거나 손님이 더 있어도 로봇이 꺼지기 때문에 로봇을 다시 시작해줘야 한다는 단점이 있었어요.

그래서 캣은 `while`문을 활용하여 영업이 종료되거나 생선이 다 팔릴 때까지 작동을 하는 로봇을 만들기로 했습니다.

우선 간단하게 `while` 문을 살펴보도록 해요.

```
#[In]

a = 1
while a < 10 :
    print(a)
    a += 1
```

Python ▾

`while`문은 조건이 참인 동안에 명령을 반복해서 수행합니다. 반복할 명령은 들여쓰기로 구분되며 조건이 거짓이면 들여쓰기로 구분되어 있는 반복 구문을 탈출합니다.

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



위의 예제에서 9를 출력하고 마지막 $a+=1$ 이 연산(할당연산)되면 $a=10$ 이 되어 조건이 거짓이 되므로 루프를 탈출 합니다. 다음 a 의 값이 10이 된다는 것 잊지 마세요.

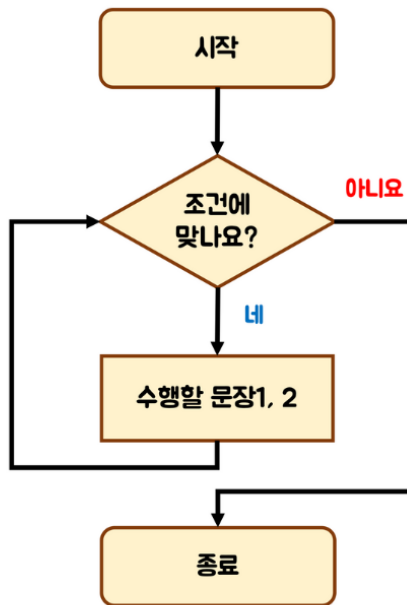
```

while문의 구조

while 조건문 :
    수행할 문장1
    수행할 문장2
    
```

Python ▾

- 반복문 순서도(Flow Chart) - while 문



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 무한반복 while, break

그럼 이제 생선이 다 팔리면 로봇이 종료되도록 만들어볼까요? 현재 고등어는 5개가 남았고, 고등어가 0개가 된다면 다 팔았다는 말과 함께 while문을 종료하고 싶어요.

```
#[In]

fish = 5
while True:
    print("고등어 ", fish, "개 남았습니다.")
    fish -= 1
    if fish == 0:
        print("고등어 다 팔렸습니다.")
        break
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

고등어 5 개 남았습니다.
고등어 4 개 남았습니다.
고등어 3 개 남았습니다.
고등어 2 개 남았습니다.
고등어 1 개 남았습니다.
고등어 다 팔렸습니다.
```

Python ▾

while 문의 조건을 부분에 bool형인 True가 오게 하면 반복문은 무한 반복되게 됩니다. 무한 반복이 일어나면 프로그램이나 서비스가 죽는 사태가 발생되기 때문에 조건을 탈출할 수 있는 구문을 중간에 입력하거나 메모리 공간을 확보할 수는 시간을 주어야 합니다. 여기서는 break 문으로 조건을 탈출하도록 설계하였습니다.

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



2.2 while, else

위 예시와 똑같은 문장을 else문으로도 만들어 볼 수 있어요. 여기서 주석 처리된 부분을 풀어보시고 주석이 될 때와 되지 않을 때를 비교해보세요.

Type '/' for commands

```
#[In]

fish = 5
while fish > 0:
    print("고등어 ", fish, "개 남았습니다.")
    # if fish < 3:
    #     break
    fish -= 1
else:
    print('고등어 다 팔렸습니다.')
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

고등어 5 개 남았습니다.
고등어 4 개 남았습니다.
고등어 3 개 남았습니다.
고등어 2 개 남았습니다.
고등어 1 개 남았습니다.
고등어 다 팔렸습니다.
```

Python ▾

else문은 다양하게 활용(특히 if문에서)됩니다. while문에서 else는 루프가 정상 종료되었을 때와 처음부터 while의 조건문이 False일 경우 실행됩니다. 위 두번째 예제에서 a==5인 경우 break문을 만나면 else 문을 실행하지 않고 빠져나옵니다.

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. break

break는 흐름을 끊어 중단할 때 사용합니다. 특정 코드를 반복하고자 for, while 문을 이용하여 반복문(loop)을 만들었다면 break 문을 사용하여 반복문을 중단할 수 있습니다. **여기서 주의하셔야 할 점은 이 break 구문은 바로 위의 for나 while문만 탈출한다는 것입니다!**

반복문(loop) 순회 중에 break 문을 만나면 반복문의 내부 블록을 벗어나게 됩니다. 그러나 예제 1번과 같이 어떠한 장치 없이 break를 넣으면 반복문이 반복하지 않으므로 예제 2번과 같이 주로 조건 안에 넣어 실행합니다.

```
# 예제1(while)
while 조건:
    반복 실행할 코드
    break # while 구문을 탈출

# 예제1(for)
for 변수 in 범위:
    break # for 반복문 탈출
```

Python ▾

```
# 예제2(while)
while 조건:
    반복 실행할 코드
    if 조건:
        break # while 구문을 탈출

# 예제2(for)
for 변수 in 범위:
    반복 실행할 코드
    if 조건 :
        break # for 반복문 탈출
```

Python ▾

Project name :

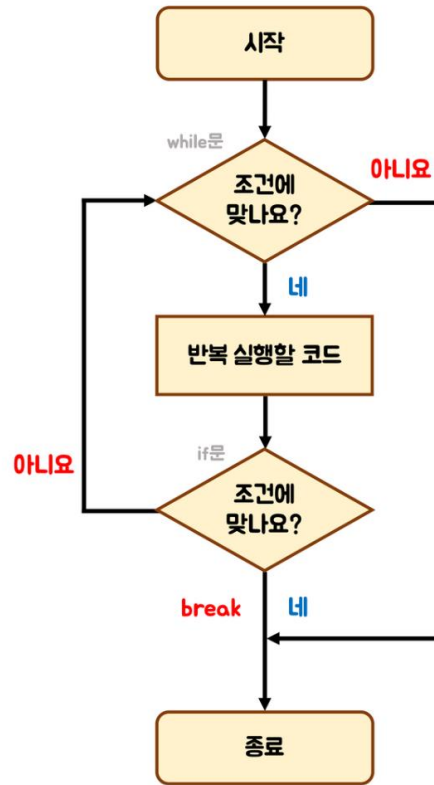
DATE

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



- 순서도(Flow Chart) - while, break 문



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



- 순서도(Flow Chart) - for , break 문



3.1 break 문

```

#[In]

시간 = 0
현재시간 = int(input("시(hour)를 입력해주세요. "))
종료시간 = 18
if 현재시간 < 종료시간:
    print('영업시간이 끝났기 때문에 영업을 종료합니다.')
else:
    while True:
        시간 += 1
        현재시간 += 1
        print('{}시간이 지나 {}시가 되었습니다'.format(시간, 현재시간))
        if 현재시간 == 종료시간:
            print('영업 종료시간이 되었기 때문에 영업을 종료합니다.')
            break
    
```

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]

시(hour)를 입력해주세요. 10
1시간이 지나 11시가 되었습니다
2시간이 지나 12시가 되었습니다
3시간이 지나 13시가 되었습니다
4시간이 지나 14시가 되었습니다
5시간이 지나 15시가 되었습니다
6시간이 지나 16시가 되었습니다
7시간이 지나 17시가 되었습니다
8시간이 지나 18시가 되었습니다
영업 종료시간이 되었기 때문에 영업을 종료합니다.
```

Python ▾

반복문이 실행되다 시간이 지나 현재 시간과 종료 시간이 같아지는 경우 영업을 종료한다는 메시지를 출력 후 `break` 문을 통하여 반복문을 빠져나갑니다.

반복문을 빠져나가자 반복문이 중단되어 더 이상 실행되지 않으므로 시간이 지나가지 않습니다.

앞서 말씀드린 것처럼, 중첩 반복문에서 바로 위에 반복문만 탈출하는 경우를 살펴보도록 하겠습니다. 아래의 경우 각 단의 `i x 5` 까지만 실행이 됩니다.

```
#[In]

for i in range(2, 10):
    for j in range(1, 10):
        if j > 5:
            break
        print(f'{i} X {j} = {i*j}')
```

Python ▾

4. continue, pass

`continue`의 사전적 의미를 살펴보면 '계속하다'라는 뜻이 있습니다. 파이썬에서 `continue`문은 반복문이 실행하는 코드 블록의 나머지 부분을 실행하지 않고 다음 반복으로 건너가게 흐름을 조정합니다.

`pass`의 사전적 의미는 '지나치다'라는 뜻이며, 파이썬에서 `pass`문은 단순히 실행할 코드가 없다는 것을 의미하며 아무런 동작을 하지 않고 다음 코드를 실행합니다.

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



continue와 pass는 구분해서 사용을 해야 하니 차이점을 잘 정리해 두세요.

반복문 순회 도중 continue문을 만날 시 continue문은 이후의 나머지 반복문 내부 코드 블록을 실행하지 않고 다음 아이템을 선택하여 반복문의 내부 코드 블록 시작 부분으로 이동합니다.

break와 마찬가지로 예제1과 같이 조건이 없이 continue를 사용하는 것은 큰 의미가 없으므로 주로 예제2처럼 조건문을 달아 사용합니다.

```
# 예제1(while)
while 조건:
    반복 실행할 코드
    continue #while 다음 반복문 수행
    반복 실행할 코드 # continue 사용시 무시되는 코드

# 예제2(for)
for 변수 in 범위:
    반복 실행할 코드
    continue #for 다음 반복문 수행
    반복 실행할 코드 # continue 사용시 무시되는 코드
```

Python ▾

```
# 예제1(while)
while 조건:
    반복 실행할 코드
    if 조건 :
        continue #while 다음 반복문 수행
    반복 실행할 코드 # continue 사용시 무시되는 코드

# 예제2(for)
for 변수 in 범위:
    반복 실행할 코드
    if 조건:
        continue # for 다음 반복문 수행
    반복 실행할 코드 # continue 사용시 무시되는 코드
```

Python ▾

Project name :

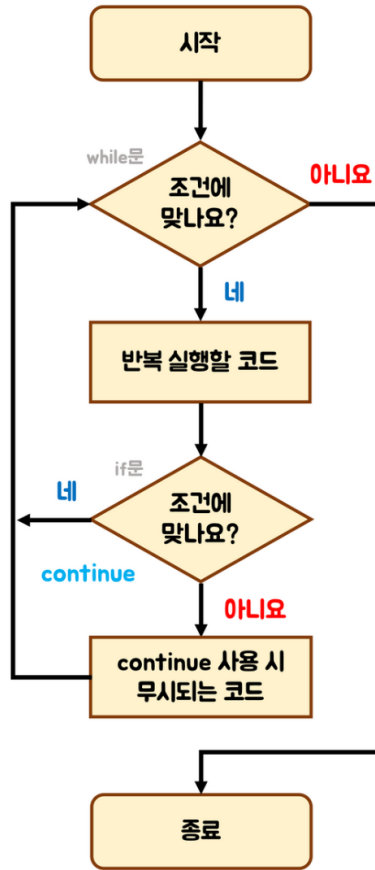
DATE

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



- 순서도(Flow Chart) - while, continue 문



Project name :

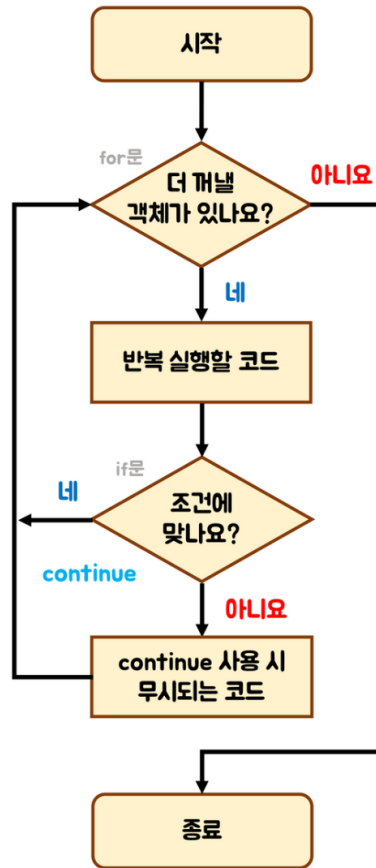
DATE

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



- 순서도(Flow Chart) - for, continue 문



```

#[In]

#continue
구매개수_총가격= [
    (3, 15000),
    (5, 25000),
    (1, 5000),
    (8, 40000),
    (0, 0),
    (2, 10000)
]

for 구매개수, 총가격 in 구매개수_총가격:
    if 구매개수 < 1:
        continue
    print('구매개수는 {}개이며 {}원입니다.'.format(구매개수, 총가격))
    
```

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]
```

```
구매개수는 3개이며 15000원입니다.
구매개수는 5개이며 25000원입니다.
구매개수는 1개이며 5000원입니다.
구매개수는 8개이며 40000원입니다.
구매개수는 2개이며 10000원입니다.
```

Python ▾

1개도 구매하지 않았을 경우 if의 조건이 참(True)이 되어 continue문을 실행합니다.

따라서 나머지 코드블록인 구매개수와 가격을 출력해주는 print문을 실행하지않고 다시 **for 반복문의 내부 코드블록 시작부분**으로 돌아갑니다.

여기서 continue라고 되어 있는 부분을 pass로 바꾸면 수험번호 1005번의 58점도 출력하는 것을 볼 수 있습니다. 반복 중간에 continue를 만나면 다음 반복으로 넘어가지만 pass를 만나면 그 라인만 지나치고 아래 문장을 그대로 실행합니다.

```
#[In]
```

```
#pass
```

```
구매개수_총가격 = [
    (3, 15000),
    (5, 25000),
    (1, 5000),
    (8, 40000),
    (0, 0),
    (2, 10000)
]
```

```
for 구매개수, 총가격 in 구매개수_총가격:
```

```
    if 구매개수 < 1:
```

```
        pass
```

```
    print('구매개수는 {}개이며 {}원입니다.'.format(구매개수, 총가격))
```

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]

구매개수는 3개이며 15000원입니다.
구매개수는 5개이며 25000원입니다.
구매개수는 1개이며 5000원입니다.
구매개수는 8개이며 40000원입니다.
구매개수는 0개이며 0원입니다.
구매개수는 2개이며 10000원입니다.
```

Python ▾

5. else

앞서 살펴본 것처럼 Python에서는 while, for 문에서도 else문을 사용할 수 있습니다. 여기서의 else는 if에서의 else처럼 '그렇지 않으면'이라는 의미 보다는 '그런 다음'이라는 의미가 더 강하기 때문에 then으로 쓰여야 된다는 논의가 있기도 했습니다.

- else문 순서도(Flow Chart) - while문



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



- else문 순서도(Flow Chart) -for문



반복문이 break 등에 의해 중단없이 정상적으로 반복이 종료된 후 특정 코드를 실행하게 해야할 때 while~else, for~else를 사용할 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



```

while 조건:
    반복 실행할 코드
else:
    while 반복문이 모두 실행되어 종료되고 실행할 코드

```

Python ▾

```

for 변수 in 범위:
    반복 실행할 코드
else:
    for 반복문이 모두 실행되어 종료되고 실행할 코드

```

Python ▾

```

#[In]

#for, else문
for i in range(5, 0, -1):
    print("고등어 ", i, " 개 남았습니다.")
else:
    print('고등어 다 팔렸습니다.')

```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```

#[Out]

고등어 5개 남았습니다.
고등어 4개 남았습니다.
고등어 3개 남았습니다.
고등어 2개 남았습니다.
고등어 1개 남았습니다.
고등어 다 팔렸습니다.

```

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



```
#[In]

#break문이 있을 때
for i in range(5, 0, -1):
    print("고등어 ", i, " 개 남았습니다.")
    if i == 1:
        break
else:
    print('고등어 다 팔렸습니다.')
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

고등어 5개 남았습니다.
고등어 4개 남았습니다.
고등어 3개 남았습니다.
고등어 2개 남았습니다.
고등어 1개 남았습니다.
```

Python ▾

위 코드에서 1번 예제처럼 break문 없이 정상 종료되었을 때에는 else문이 실행됩니다. 그러나 2번 예제처럼 break문이 있을 경우에는 else문을 실행하지 않게 됩니다.

if문에서 else는 '그렇지 않으면'이라는 의미로 조건이 거짓일 때 씁니다.

```
if 조건:
    조건이 참일 경우 실행문
else:
    조건이 거짓 일 경우 실행문
```

Python ▾

```
#[In]

i = 2
if i < 4:
    print('i는 4보다 작습니다')
else:
    print("i는 4보다 크거나 같습니다")
```

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]
i는 4보다 작습니다
```

Python ▾

try~except~else문에서 else는 예외가 발생하지 않을 때 쓰입니다.

```
try:
    실행문
except:
    예외 발생 시 처리문
else:
    예외 발생하지 않을 경우 실행문
```

Python ▾

```
#[In]

try:
    i = 1
    j = 1
    x = i/j
except:
    print("error")
else:
    print(x)
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]
1.0
```

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



6. 중첩 반복문

반복문 내부에 또 다른 반복문이 있을 경우, **중첩되었다**고 얘기합니다. 중첩 반복문이란 반복문 안에 반복문이 들어가 중첩된 것을 얘기합니다. 중첩 반복문에서는 종료되는 값을 항상 확인하세요.

```
#[In]

i = 2
while i < 10 :
    k = 1
    while k < 10:
        print(i, " * ", k, " = ", i * k)
        k += 1
    i += 1
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
3 * 1 = 3
...
...
...
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
```

Python ▾

구구단은 2단부터 시작하기에 변수 `i`를 2로 설정해줍니다. 9단을 수행한 후 종료할 수 있도록 10미만이라는 조건을 설정해주었습니다. `i`는 구구단의 단을, `k`는 순차적으로 반복될 곱의 수이며 `while`문을 하나 더 이용하여 반복될 곱의 수를 1씩 순차적으로 더해주었습니다.

여기서 중요한 것은 **종료되는 값**입니다. `k`가 10이 된 상태로 안에 반복문이 종료되기 때문에 3단을 하기 위해서는 **다시 `k`를 1로 초기화 시켜주어야 합니다.**

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



for문과 range 함수를 이용하면 좀 더 간단하게 구구단을 출력할 수 있습니다.

```
#[In]

for i in range(2,10):
    print("---{0}단---".format(i))
    for k in range(1,10):
        print(i,"*",k,"=",i*k)
```

Python ▾

위 코드를 실행시키면 아래와 같이 출력됩니다.

```
#[Out]

---2단---
2 * 1 = 2
2 * 2 = 4
2 * 3 = 6
2 * 4 = 8
2 * 5 = 10
2 * 6 = 12
2 * 7 = 14
2 * 8 = 16
2 * 9 = 18
---3단---
3 * 1 = 3
3 * 2 = 6
...
...
중략
```

Python ▾

for문을 중첩 사용하면 좀 더 간결하고 직관적인 코드를 작성할 수 있습니다.

첫 번째 for문에서 range 함수를 이용하여 2단부터 10단까지 차례로 i에 대입됩니다. 두 번째 for문에서 곱해지는 수를 k에 넣고 구구단을 출력하고 있습니다.

while과 달리 k의 값이 for문 안에서 자동으로 초기화가 되기 때문에 while문처럼 k=1을 두 번째 for문 위에 넣을 필요는 없습니다.

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



▶ 퀴즈!

생선의 종류는 고등어, 연어, 송어가 있습니다. 고등어는 5천원, 연어는 8천원, 송어는 3천원입니다. 모든 손님들은 3가지 종류의 생선을 모두 구매할 예정입니다.

5명의 손님께 구매할 생선들의 갯수를 각각 물어본 뒤, 맨 마지막엔 계산한 **총 금액**을 출력해주세요.

(문제가 어려우니 반드시 여러번 생각한 후 풀어보세요. - 힌트 : list를 사용해보세요)

#퀴즈 - 출력해야 하는 값 입니다.

고등어 몇 개를 구매하실 건가요?3

연어 몇 개를 구매하실 건가요?4

송어 몇 개를 구매하실 건가요?2

53000 원 입니다.

고등어 몇 개를 구매하실 건가요?1

연어 몇 개를 구매하실 건가요?2

송어 몇 개를 구매하실 건가요?3

83000 원 입니다.

...

종락

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





사자탈을 쓴 캣

기업에 대한 노하우가 쌓이고 곳곳에 배치할 로봇까지 만들어, 직원들에게 높은 임금과 자유를 주게 된 캣은 고민에 빠졌어요.

"이렇게 벌여 병원을 세우려면 1억 3천 299년이 걸린다냥.."

평민들만 있는 곳에서는 벌 수 있는 돈이 한계가 있었기 때문이죠. 그래서 캣은 모든 지분을 사회에 환원하고 사자들만 들어갈 수 있는 라이언 타운에 들어가기로 결심합니다.

그러나 문제가 있었어요. 라이언 타운은 왕족과 귀족인 사자들만 들어갈 수 있었으며 고양이는 사업은 커녕 라이언 타운에 들어갈 수조차 없었어요.

한참을 고민하던 캣은 아주 기발한 생각을 하게 되었습니다.



"사자탈을 쓰고 들어가면 된다냥!"

그렇게 아주 간단한 방법으로 라이언 타운에 들어갈 수 있을 줄 알았지만 생각보다 관문을 지키는 문지기들은 그리 어리숙하지 않았어요. 꼼꼼하고 살벌한 검문검색에 캣은 입장 시도조차 못하고 고민에 다시 빠졌습니다.

그때, 지분을 주었던 한 직원이 찾아와 고맙다며 정보를 주었어요!

"라이언 타운의 왕족이 은밀하게 출입을 하던 비밀통로가 있는데 그곳에 오랜 문지기가 우리 할아버지였다냥! 그곳만 통과하면, 라이언 타운 안에서는 검문 검색을 안하니 그 옷을 입고 다닐 수 있을것이다냥!"

캣은 은밀한 곳에 숨겨진 비밀통로를 찾아갔습니다. 약속대로 할아버지 문지기는 없었어요. 그런데! 생각지도 못한 난제를 만났어요. 비밀통로를 지키는 '말하는 문'을 만난거죠!



"이 문은 왕의 혈통만이 지나갈 수 있는 길! 혈통을 검증하기 위한 문제를 맞춰야 문을 열어줄 수 있다!! 크르릉!!"

라이언 타운을 지키는 '말하는 문'은 문제를 내기 시작했어요.

1. 왕의 충성스런 신하는 아래와 같다. 왕의 신하가 총 몇 명인지 구하라.

```
신하 = {
    '하이에나' : 1121,
    '코뿔소' : 122,
    '코끼리' : 88,
    '기린' : 119,
    '독수리' : 62,
    '고양이' : 31,
}
```

2. 이웃 왕국의 침략이 있어 출전 준비해야 한다. 100명 이상인 종은 출전을! 100명 이하인 종은 왕국을 수호한다. 출전 신하는 몇 명이고, 왕국을 수호하는 신하는 몇 명인가?

3. 침략을 방어하고, 출전 했던 신하들은 각각 80%라는 큰 손실을 입었습니다. 왕국을 수호하는 신하까지 총 얼마의 신하가 남아 있나요?

Python ▾

과연 캣은 문제를 풀고 라이언 타운에 들어갈 수 있을까요?

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





스토리 : 쏘아진 화살

드디어 라이언 타운 잠입에 성공한 캣! 캣은 도와줄 친구도, 동료도, 가족도, 친척도 없었지만 그동안 쌓아왔던 경험으로 라이언 타운에서도 굳게일학이었어요.

최대한 많은 물고기를 잡아 잠입한 통로를 통해 물고기를 들여왔습니다. 또 신뢰를 쌓고, 동료를 모으고, 회사를 성장시키기까지 이미 많은 경험을 통해 쌓은 캣의 기술로 마치 하늘에 쏘아진 화살처럼 회사를 성장시켰습니다.

"이제 병원을 세워야 겠다냥!"

캣은 처음 그의 뜻대로 병원을 세우기 위해 발품을 팔았습니다. 부지를 매입하고, 의사를 설득하고, 건축 설계, 법인 설립, 병원 등록까지 막히는 것이 없이 처리해 나갔어요. 그러나 문제는 전혀 생각치 못한 곳에서 터져나왔습니다.

"평민도 들어올 수 있는 병원이라니!? 그런 병원은 허락할 수 없흥! 크르릉!"

귀족들의 반발이 만만치 않았던 것이죠. 캣은 오랫동안 그들을 설득했습니다. 정치적 힘을 얻기 위해 어쩔 수 없이 정계에도 발을 들여놓았어요.



"차별은 두렵고도 무섭다냥, 파괴적 혁신이 필요하다냥."

캣은 캣의 뜻대로 병원을 설립하고, 그곳에서 평민을 받을 수 있게 되었지만, 그가 꿈꾸던 보육원, 교육 시설과 같은 차별없이 사는 사회는 멀게만 느껴졌습니다.

"방법을 찾아야 한다냥!"

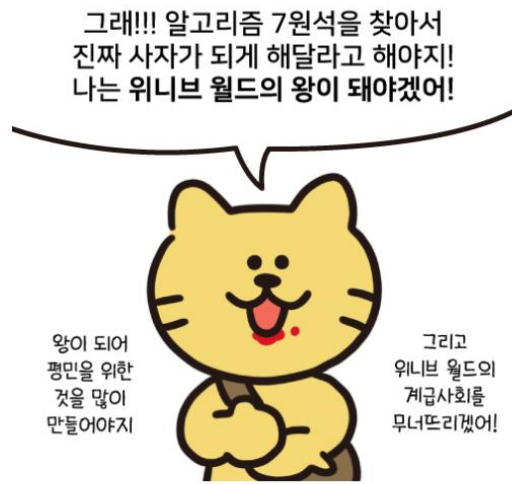
그는 어느날, 그동안 쌓았던 모든 것을 내려놓고 홀연히 사라집니다.



6편 : 파이와 썬의 알고리즘 7원석을 찾아서

1. 라이캣의 모형
2. 클래스
 - 2.1 클래스 변수와 인스턴스 변수
 - 2.2 `_init_` 함수
 - 2.3 상속
 - 2.4 다중 상속
 - 2.5 특별 메소드(magic method)
 - 2.6 캣의 준비물

1. 라이캣의 모험



유니브 월드에서는 '알고리즘 7원석'에 대한 이야기로 떠들썩합니다. 알고리즘의 제왕 '파이'와 '썬'이 어딘가에 숨겨둔 알고리즘 7원석을 찾으면 소원을 이뤄준다는 소문이 있었어요. 수행을 다니던 캣은 생각합니다.

"나는 왕이 되어야겠다냥!"

캣은 멀고 험한 길이 될 것임을 직감했습니다. 그동안 쌓았던 모든 지식이 무용지물이 될 수 있겠다는 생각을 하게 되었고, 새로운 준비를 결심했습니다.

"험한 길이 될거냥. 같으니 단단히 준비해야겠다냥!"

캣은 우선 자신이 지금까지 일구어 놓은 유니브 월드에 **회사들을 경영할 대리인**을 로봇으로 만드기로 합니다. 지금까지 했던 방식과는 다른 전혀 새로운 방식으로요.

- 라이캣의 상태창!

```

#[In]

이름 = '캣'
설명 = '위니브 월드에서 가장 성공한 사업가 라이캣, 현재 어디있는지 알 수 없음.'
나이 = 21
오늘_잡은_물고기 = '9999999'
직원수 = 124
키 = '46.1cm'
몸무게 = 1.8
잡식 = True
돈 = 9999999
정치 = 21
야망 = 2
훈장 = [
    '백상아리를 잡은 고양이',
    '성실한 납세자',
    '해골섬 낚시꾼',
    '청년 고용 착한 기업',
    '회사를 설립한 자',
    '혈통의 인정을 받은 자',
    '천민을 위한 병원을 세운 자'
]
기술 = {
    '회사운영': 99,
    '낚시': 96,
    '통발': 93,
    '큰그물': 95,
    '재무': 34,
}

```

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. 클래스

클래스는 데이터와 기능을 가지고 있는 인스턴트 객체를 생성하기 위한 역할을 합니다.

파이썬은 대표적인 객체지향 프로그래밍 언어입니다. 클래스는 일종의 설계도면입니다. 파이썬은 이 설계도면을 보며 하나의 인스턴스 객체를 만들어 냅니다. 그리고 그 인스턴스 객체를 사용할 수 있게 됩니다.

어렵죠? 기억하세요. 지금 컷은 자신을 대리할 대리인을 만드는 것입니다. 따라서, **클래스**는 대리인을 찍어내는 **공장!** 그 공장에서 찍혀져 나오는 **인스턴스** 로봇이 바로 컷과 똑같은 생각을 하고 똑같이 행동하는 **대리인**입니다.

```
#[In]

class 대리인(object):
    이름 = '라이캣의 대리인'
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')

대리인1 = 대리인()
대리인2 = 대리인()

대리인1.계산하기(5)
대리인1.포장하기()
대리인2.계산하기(10)
대리인2.포장하기()

print(대리인1.이름)
print(대리인1.훈장)

print(대리인2.이름)
print(대리인2.훈장)
```

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



```
#[Out]
```

```
25000 원 입니다.
포장이 완료되었습니다.
50000 원 입니다.
포장이 완료되었습니다.
```

```
라이캣의 대리인
```

```
['백상아리를 잡은 고양이', '성실한 납세자', '해골섬 낚시꾼', '청년 고용 착한 기업', '회사를 설립한 자', '혈통의 인정을 받은 자', '천민을 위한 병원을 세운 자']
```

```
라이캣의 대리인
```

```
['백상아리를 잡은 고양이', '성실한 납세자', '해골섬 낚시꾼', '청년 고용 착한 기업', '회사를 설립한 자', '혈통의 인정을 받은 자', '천민을 위한 병원을 세운 자']
```

Python ▾

그런데 대리인이 이름도 같고 훈장도 같으니, 구분이 안되죠? 그래서 대리인 목록을 하나 만들도록 하겠습니다. 또한 이름이 모두 같으니 이름도 다르게 하겠습니다.

2.1 클래스 변수와 인스턴스 변수

```
#[In]
```

```
class 대리인(object):
    # 클래스 변수 위치 (파이썬 규약에 따라 indent로 결정)
    이름 = '라이캣의 대리인'
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')
```

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



```

def 포장하기(self):
    print('포장이 완료되었습니다.')

대리인1 = 대리인()
대리인.대리인_목록.append('대리인1')
대리인2 = 대리인()
대리인.대리인_목록.append('대리인2')

print(대리인.대리인_목록)
print(대리인1.대리인_목록)
print(대리인2.대리인_목록)

```

Python ▾

```

#[Out]

['대리인1', '대리인2']
['대리인1', '대리인2']
['대리인1', '대리인2']

```

Python ▾

클래스 변수는 다른 인스턴스들과 변수를 공유합니다. 이 변수는 메서드(클래스의 함수라고는 부르지 않습니다)의 위치와 동등한 들여쓰기 위치에 자리하고 있습니다.

이 클래스 변수는 위의 예시와 같이 인스턴스나 클래스 이름을 통해서 접근할 수 있습니다. 변수 이름으로 직접 접근을 하지 않는다는 사실을 주의해주세요!

이 클래스 변수는 해당 클래스를 통해 만들어진 모든 인스턴스 객체들이 공유하는 변수 값입니다. 각 인스턴스 객체들 각자가 관리하고 있는 변수는 인스턴스 변수라고 합니다.

위의 예시에서는 인스턴스를 생성할 때마다 어떤 인스턴스가 만들어졌는지 확인하기 위해 인스턴스를 생성할 때마다 `대리인.대리인_목록` 에 접근하여 해당 리스트에 어떤 대리인이 추가되었는지를 표시하고 있습니다.

하지만 이렇게 처음 시작 시에 일일이 메서드를 호출해서 하는 것은 불편하고 실수가 생길 수 밖에 없는 작업입니다. 이런 불편을 해소하기 위해 `__init__` 메서드가 있습니다.

언더바가 두개인 이 메서드는 매직 메서드 또는 던더함수라고 합니다. 이것은 다음 장에서 알아보도록 하겠습니다.

이번 장에서는 인스턴스 변수를 통해, 이름을 변경해보죠!

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



```

#[In]

class 대리인(object):
    이름 = '라이캣의 대리인'
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def 이름_변경하기(self, name):
        self.이름 = name

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')

대리인1 = 대리인()
대리인.대리인_목록.append('대리인1')
대리인2 = 대리인()
대리인.대리인_목록.append('대리인2')

print(대리인1.이름)
print(대리인2.이름)

대리인1.이름_변경하기('라이캣하나')
대리인2.이름_변경하기('라이캣둘')

print(대리인1.이름)
print(대리인2.이름)

```

Python ▾

```

#[Out]

라이캣의 대리인
라이캣의 대리인
라이캣하나
라이캣둘

```

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



클래스 변수를 다룰 때는 클래스 자체(예시에서는 **대리인** 객체)를 이용하여 다루어야 하지만 위의 예시에서 이름을 변경할 때는 각 인스턴스 객체들이 각각 자신의 메서드를 통해 접근을 하고 있습니다. 여기서 `self`의 정체는 바로, 인스턴스의 영역을 얘기합니다.

앞서 **대리인 목록**을 모두가 공유했다면, `self`의 영역은 다른 인스턴스에 해당 변수를 공유하지 않는 **고유 영역**이라 볼 수 있습니다.

2.2 __init__ 함수

자, 이제 `__init__`을 사용해서, 이름 변경하기를 별도의 메서드가 아닌, 인스턴스가 생성될 때 호출이 되게 해주도록 하겠습니다.

```
[In]

class 대리인(object):
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def __init__(self, name):
        self.대리인_목록.append(name)
        self.이름 = name

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')

대리인1 = 대리인('라이캣하나')
대리인2 = 대리인('라이캣둘')

print(대리인1.이름)
print(대리인2.이름)

print(대리인1.대리인_목록)
print(대리인2.대리인_목록)
```

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



```
#[Out]

라이캣하나
라이캣둘
['라이캣하나', '라이캣둘']
['라이캣하나', '라이캣둘']
```

Python ▾

이 메서드는 인스턴스 객체 생성 시 자동으로 실행합니다. 어떤가요? 코드가 좀 더 간결해졌습니다!
좀 더 알아보자면 `__init__` 메서드는 다른 프로그래밍 언어에서의 생성자(`constructor`) 역할을 하는 클래스 메서드입니다.

2.3 상속

```
#[In]

class 대리인(object):
    이름 = '라이캣의 대리인'
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def __init__(self, name):
        self.대리인_목록.append(name)
        self.이름 = name

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')
```

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



```
class 진화된_대리인(대리인):
    #이제 유니브 월드에 있는 대리인들 끼리는 서로 협업합니다.
    추가_서로협업하기 = True

    def 추가된_기능_청소기능(self):
        print("청소기능이 추가되었습니다.")

대리인1 = 대리인('라이캣하나')
대리인2 = 진화된_대리인('라이캣둘')

대리인2.추가된_기능_청소기능()
대리인2.이름
대리인2.훈장
```

Python ▾

```
#[Out]

청소기능이 추가되었습니다.
None
라이캣둘
['백상아리를 잡은 고양이',
 '성실한 납세자',
 '해골섬 낚시꾼',
 '청년 고용 착한 기업',
 '회사를 설립한 자',
 '혈통의 인정을 받은 자',
 '천민을 위한 병원을 세운 자']
```

Python ▾

클래스를 상속하는 방법은 위의 예시처럼 새로운 클래스의 `()` 안에 상속할 클래스명을 적어주는 것입니다.

`진화된_대리인` 클래스는 기존의 `대리인` 클래스가 사용하던 데이터와 기능들을 모두 사용할 수 있으며, 추가적인 기능을 만들었습니다.

또한 데이터(클래스 변수와 메서드를 재정의할 수 있습니다)를 새로 덮어씌울 수도 있으며 자기 자신만의 클래스 변수를 새로 정의할 수 있습니다.

여기에서 상속을 하는 `대리인` 을 부모 클래스(슈퍼 클래스)라고 부르고 `진화된_대리인` 을 자식 클래스(서브 클래스)라고 부릅니다.

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



2.4 다중 상속

여러 클래스를 상속받을 때는 아래와 같이 사용하면 됩니다. 다중 상속은 단일 상속 법과 같습니다. 여러개를 상속 받을 때에는 콤마를 사용합니다.

```
#[In]

class 배터리_영구_증가(object):
    pass

class 효율_극대화(object):
    pass

class 대리인(object):
    이름 = '라이캣의 대리인'
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34,
    }

    def __init__(self, name):
        self.대리인_목록.append(name)
        self.이름 = name

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')

class 진화된_대리인(대리인, 배터리_영구_증가, 효율_극대화):
    #이제 유니브 월드에 있는 대리인들 끼리는 서로 협업합니다.
    추가_서로협업하기 = True

    def 추가된_기능_청소기능(self):
        print("청소기능이 추가되었습니다.")
```

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



2.5 특별 메소드(magic method)

앞서 말씀드린 것처럼 파이썬에서의 클래스에는 기본적으로 내장하고 있는 특별 메서드들이 있습니다. 파이썬에서는 이런 내장하고 있는 특별 메서드들을 쉽게 재정의하여 사용할 수 있게 되어 있습니다. 매직 메서드라고 부릅니다. 혹은 더블 언더바(__)를 쓰고 있는 점에서 줄여서 뉘더(dunder) 메서드라고도 부릅니다.

```
#[In]

class 신규_로봇(object):
    def __init__(self, name):
        self.name = name
    def __getattr__(self, item):
        print(item + '속성은 존재하지 않습니다.')

로봇1 = 신규_로봇('로봇1')
print(로봇1.name)
print(로봇1.청소하기)
```


Python ▾

```
#[Out]

로봇1
청소하기속성은 존재하지 않습니다.
None
```

Python ▾

위의 예시에서 `__init__` 과 `__getattr__` 이 특별 메서드입니다. 예시에서 보는 바와 같이 파이썬에 내장되어 있는 특별 메서드들을 쉽게 재정의하여 사용할 수 있습니다.

 특별 메서드들의 종류를 자세하게 알고 싶다면 파이썬 공식 홈페이지를 참고바랍니다.
(<https://docs.python.org/3/reference/datamodel.html>)

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



2.6 캣의 준비물

자, 그럼 앞서 `type` 함수를 호출하면 나왔던 `class 'int'` 는 무엇일까요? 맞습니다. python은 모든 자료형이 class로 되어 있어요!

위니브 월드에 비유해보자면, 사실은 실제하는 캣도, class 였던 것이죠! 이제 대리인의 type과 dir을 찍어보세요. 보다 확연히 알 수 있을 겁니다.

모든 준비가 끝났으니 가볍게 준비물을 챙겨 떠나보도록 하죠. append, insert 등의 매서드를 사용해서 필요한 준비물들을 넣어보세요. 창의력을 발휘해보세요.

여러분들이라면 무엇을 챙겨가시겠나요?

```
# 캣의 준비물
준비물 = [??, ??, ??, ??, ??]
```

Python ▾

자, 이제 제대로 된 여행준비를 하고, 여행을 떠나봅시다.



해당 스토리는 자고 일어나니 코딩테스트 시리즈와 이어집니다.



Project name :

DATE

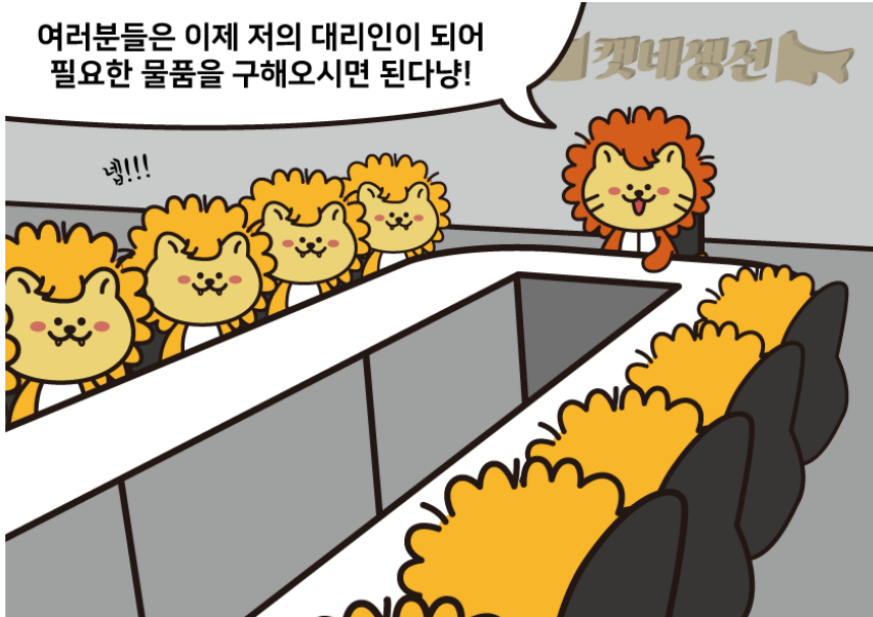
비고	Line	File name :
	1	
	2	
	3	
	4	
	5	
	6	
	7	
	8	
	9	
	10	
	11	
	12	
	13	
	14	
	15	
	16	
	17	
	18	
	19	
	20	
	21	
	22	
	23	
	24	
	25	
	26	
	27	
	28	
	29	
	30	

MEMO





갯의 모험 자금을 모아라!



라이캣 대리인 10명을 만들어 여행에 필요한 물품을 알아서 구해오도록 시킬 생각입니다. 물론 자신도 직접 뛰어 물품을 구할 생각이예요.

"물론 혼자서도 할 수 있지만 기회비용을 생각했을 때, 최선의 결과를 도출하기 위해 대리인을 만드는 것이 좋겠다냥!"

라이캣은 각각의 대리인들이 비교우위를 점하지 않기 때문에 일을 적절하게 분배하기로 했습니다.

1. 대리인 10명을 생성하세요!

아래 코드를 참고하여 물건을 구할 대리인을 설계하십시오.

```
class 대리인(object):
    pass

대리인_그룹 = []
for name in range(10):
    대리인_그룹.append(대리인(str(name)))
```

Python ▾

→ 대리인의 이름은 0, 1, 2, 3, 4...9로 10명입니다!

→ 경비를 구하는 기능이 있습니다. 아래와 같이 입력했을 때, 시간당 100만 노드를 버는 메서드를 만들어주세요.

```
인스턴스이름.경비구하기('10시간')
```

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



→ 물건을 구하는 기능이 있습니다. 아래와 같이 입력을 했을 때, 시간당 필요 물품 1개를 구해오는 메서드를 만들어주세요. 물품은 클래스 변수로, 모두가 공유합니다.

```
#[In]
인스턴스이름.물품구하기('3시간')
인스턴스이름.물품

#[Out]
['대리인이 구해온 물품 1', '대리인이 구해온 물품 2', '대리인이 구해온 물품 3']
```

Python ▾

2. 대리인에게 스킬 추가하기!

아래와 같은 코드가 있었다고 하였을 때 이 대리인의 모든 스킬 합을 구하십시오. 또, 스킬을 추가해주는 메서드를 만들어주세요!

```
class 대리인(object):
    이름 = '라이캣의 대리인'
    대리인_목록 = []
    훈장 = [
        '백상아리를 잡은 고양이',
        '성실한 납세자',
        '해골섬 낚시꾼',
        '청년 고용 착한 기업',
        '회사를 설립한 자',
        '혈통의 인정을 받은 자',
        '천민을 위한 병원을 세운 자'
    ]
    기술 = {
        '회사운영': 99,
        '낚시' : 96,
        '통발' : 93,
        '큰그물' : 95,
        '재무' : 34
    }

    def __init__(self, name):
        self.대리인_목록.append(name)
        self.이름 = name

    def 계산하기(self, x):
        print(int(x)*5000, '원 입니다.')

    def 포장하기(self):
        print('포장이 완료되었습니다.')
```

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. 진화된 대리인!

라이캣은 자신과 같이 일하는 좀 더 진화된 대리인을 원했습니다. 아래 기능을 도입하여 좀 더 진화된 대리인을 만들어 봅시다. 각각에 메서드는 자유롭게 구현해보세요.

- 진화된_대리인1은 던전 탐험 추가, 희귀 아이템을 수집합니다.

- 진화된_대리인2는 위험 요소를 미리 판단하여 유사 시 자금을 회피할 수 있는 기능을 도입합니다.

```
진화된_대리인1.던전탐험('진홍의 목걸이')
진화된_대리인2.위험판단('위험')
진화된_대리인2.위험판단('평범')
진화된_대리인2.위험판단('긴장감 교조')
```

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





파이와 썬이 남긴 단 하나의 단서

모든 알고리즘을 해독할 수 있는 알고리즘 7 원석을 보유한 알고리즘 제왕 파이와 썬은 죽기 전, 이 보물에 '암호'를 걸어 세계 어딘가에 묻어놨다고 공표하였다. 그가 남긴 문자는 아래와 같다.

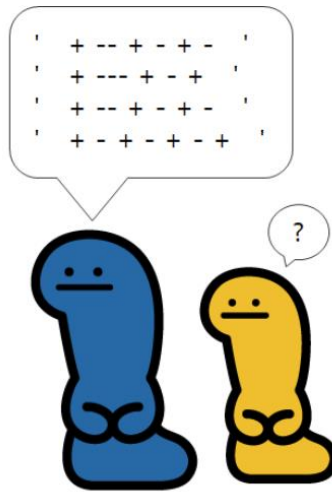
섬으로 향하라!

```

' + - - + - + - '
' + - - - + - + '
' + - - + - + - '
' + - + - + - + '
    
```

해(1)와 달(0),
Code의 세상 안으로! (En-Coding)

Python ▾



출력조건 : 문자열

해당 문제는 눈떠보니 코딩 테스트 전날로 이어집니다. Bootcamp 알은물, 고생 많으셨어요! 😊

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



초판 1쇄 발행 | 2020년 7월 6일

지은이 | 이호준 김혜원 김유진 차경림 김난화 김승민 김진 이송신 이진우

편 집 | 이호준

총 괄 | 이호준

펴낸곳 | 사도출판

주 소 | 제주특별자치도 제주시 동광로 137 대동빌딩 4층

표지디자인 | 차경림

홈페이지 | <http://www.paullab.co.kr>

E-mail | paul-lab@naver.com

ISBN | 979-11-88786-33-6

Copy right © 2020 by. 사도출판

이 책의 저작권은 사도출판에 있습니다. 저작권법에 의해 보호를 받는 저작물이므로 무단 복제 및 무단 전재를 금합니다.

이 책에 대한 의견을 주시거나 오타자 및 잘못된 내용의 수정 정보는 사도출판의 이메일로 연락을 주시기 바랍니다.



PYTHON BOOTCAMP

비매품/무료

95560



9 791188 786336

ISBN 979-11-88786-33-6 (PDF)