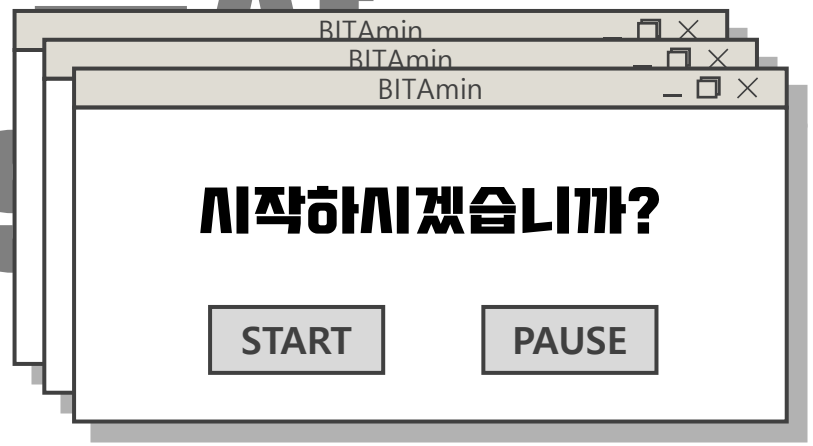
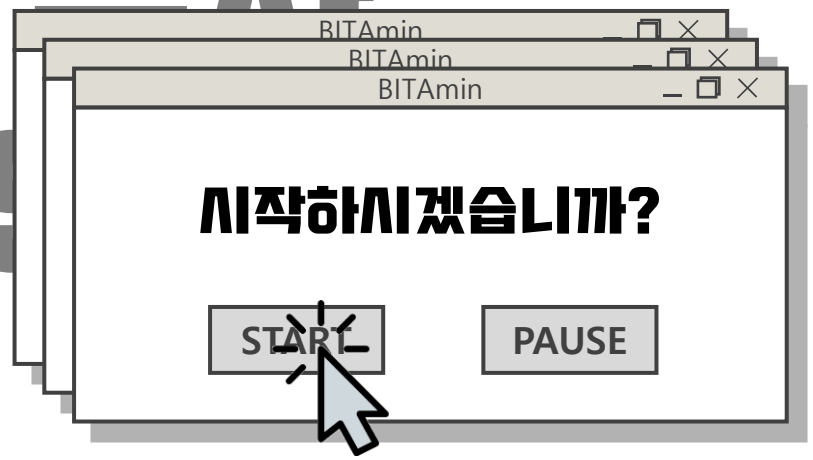
Gram

1주차
SESSION

1주차
SES



1주차
SES



목차

1. 조건문
- 조건문이란 ?

참과 거짓을 "판단" 하는 문장

조건이 참일 때 다음 문장을 수행

1. 비교 연산자
2. 논리 연산자
3. in / not in 등

1. 조건문
- 조건문이 필요한 이유

" 비타민에 합격하면 행복하고, 떨어지면 슬프다. "

주어진 조건을 판단

해당 조건에 맞게 상황 수행

1. 조건문
- 조건문이 필요한 이유

“ 비타민에 합격하면 행복하고, 떨어지면 슬프다. ”

```
bitamin = True

if bitamin:
    print("행복하다")
else :
    print("슬프다")
```

행복하다

- bitamin에 True를 입력했으므로 bitamin은 참
- 따라서 조건문이 수행되어 ‘행복하다’ 가 출력된다

1. 조건문
- if문의 기본 구조

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else:
    수행할 문장A
    수행할 문장B
    ...
```

- If 조건문 뒤에는 항상 콜론(:)이 붙는다
- 조건문이 참이면 if문 바로 다음 문장들을 수행
- 조건문이 거짓이면 else문 다음 문장들을 수행
- else문은 if문 없이 독립적인 사용이 불가능

```
if 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
```

- if문에 속하는 모든 문장에는 꼭 들여쓰기를 해주어야 함
- 꼭 주의합시다 !!!

* 번외
- 들여쓰기(Indent)

- 가독성을 위한 것
- 영역을 지정한다는 뜻
- 문법적인 강제사항
- Spacebar를 활용한 한 칸, 두 칸, 네 칸 등의 방법이 존재
- Tab 활용도 하나의 방법
- 방법은 상관없지만 혼용은 하지 말 것 (오류의 원인이 됨)
- 요즘 파이썬 커뮤니티에서는 Spacebar 4개를 사용하는 것을 권장함

1. 조건문
- if문의 기본 구조

```
이소연 = True

if 이소연:
    print("안녕하세요")
print("저는")
    print("비타민 6기 교육부를 맡고 있습니다")
```

```
  File "<ipython-input-3-5a034a8812e8>", line 6
    print("비타민 6기 교육부를 맡고 있습니다")
    ^
IndentationError: unexpected indent
```

```
이소연 = True

if 이소연:
    print("안녕하세요")
    print("저는")
        print("비타민 6기 교육부를 맡고 있습니다")
```

```
  File "<ipython-input-4-5ad337f46722>", line 6
    print("비타민 6기 교육부를 맡고 있습니다")
    ^
IndentationError: unexpected indent
```

들여쓰기 오류 발생 !!!!!!!!!

예습과제시 꼭 올바른 코드로 작성해보세요 ~

1. 조건문
- 비교 연산자

비교연산자	설명
x < y	x가 y보다 작다
x > y	x가 y보다 크다
x == y	x와 y가 같다
x != y	x와 y가 같지 않다
x >= y	x가 y보다 크거나 같다
x <= y	x가 y보다 작거나 같다

" 만약 시간이 오후 10시가 넘었다면 참고, 그렇지 않으면 먹어라 "

```
hour = 21

if hour >= 22:
    print("참아라")
else:
    print("먹어라")
```

```
먹어라
```

- hour는 21
- hour >= 22 조건문이 거짓이 되기 때문에 else문 다음 문장을 수행

1. 조건문
- 논리 연산자

연산자	설명
x or y	x와 y 둘중에 하나만 참이어도 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

" 시간이 오후 10시가 넘었고 배가 고프지 않다면 참고,
그렇지 않으면 먹어라 "

```
hour = 23
hungry = True

if hour >= 22 and not hungry:
    print("참아라")
else:
    print("먹어라")
```

먹어라

- hungry가 True 이므로 not hungry는 False가 됨
- 오후 10시가 넘었지만 배가 고프기 때문에 else문 다음 문장을 수행

1. 조건문
- in / not in 연산자

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

“ 주머니에 돈이 있으면 떡볶이를 먹고, 없으면 먹지 마라라 ”

```
pocket = ['money', 'cellphone', 'paper']

if 'money' in pocket:
    print("떡볶이를 먹어라")
else :
    print("먹지 말아라")
```

```
떡볶이를 먹어라
```

- pocket 리스트 안에 ‘money’ 가 있으므로 ‘money’ in pocket은 참
- 따라서 if문 다음 문장이 수행

1. 조건문
- in / not in 연산자

“ 주머니에 돈이 있으면 떡볶이를 먹고, 주머니에 돈은 없지만 카드가 있으면 먹고, 돈도 없고 카드도 없으면 먹지 마라라 ”

```
pocket = ['cellphone', 'paper']
card = True

if 'money' in pocket:
    print("떡볶이를 먹어라")
else :
    if card:
        print("떡볶이를 먹어라")
    else:
        print("먹지 말아라")
```

```
떡볶이를 먹어라
```

1. 조건문
- in / not in 연산자

“ 주머니에 돈이 있으면 떡볶이를 먹고, 주머니에 돈은 없지만 카드가 있으면 먹고, 돈도 없고 카드도 없으면 먹지 마라라 ”

```
pocket = ['cellphone', 'paper']
card = True

if 'money' in pocket:
    print("떡볶이를 먹어라")
else :
    if card:
        print("떡볶이를 먹어라")
    else:
        print("먹지 말아라")
```

떡볶이를 먹어라

BITAmin

- if와 else만으로 다양한 조건 판단 까다로움
- 참과 거짓만으로 분류하는 한계
- 가독성 좋지 않음
- 좀 더 간결한 코드를 사용하자 !

1. 조건문
- elif 문

“ 주머니에 돈이 있으면 떡볶이를 먹고, 주머니에 돈은 없지만 카드가 있으면 먹고, 돈도 없고 카드도 없으면 먹지 마라라 ”

```
pocket = ['cellphone', 'paper']
card = True

if 'money' in pocket:
    print("떡볶이를 먹어라")
else :
    if card:
        print("떡볶이를 먹어라")
    else:
        print("먹지 말아라")
```

떡볶이를 먹어라

```
pocket = ['cellphone', 'paper']
card = True

if 'money' in pocket:
    print("떡볶이를 먹어라")
elif card:
    print("떡볶이를 먹어라")
else:
    print("먹지 말아라")
```

떡볶이를 먹어라

1. 조건문
- elif 문

“ 주머니에 돈이 있으면 떡볶이를 먹고, 주머니에 돈은 없지만 카드가 있으면 먹고, 돈도 없고 카드도 없으면 먹지 마라라 ”

```
pocket = ['cellphone', 'pape
card = True

if 'money' in pocket:
    print("떡볶이를 먹어라")
else :
    if card:
        print("떡볶이를 먹어
    else:
        print("먹지 말아라")
```

```
떡볶이를 먹어라
```

BITAmin

- 여러가지 상황 (다중 조건) 판단이 가능
- if문 다음에 또 if문이 필요할 때 사용
- 가독성을 높임

1. 조건문
- elif 문

간단한 숫자 게임 해보기

```
num = int(input("1~10사이의 숫자를 입력해주세요: "))

if num > 5 and num <= 10:
    print('5보다 큽니다')
elif num > 0 and num <= 5:
    print('5보다 작거나 같습니다')
else :
    print('범위를 벗어납니다')
```

```
1~10사이의 숫자를 입력해주세요: 11
범위를 벗어납니다
```

간단한 음료수 자판기 만들어 보기

```
button = int(input("버튼을 눌러주세요: "))

if button == 1:
    print('콜라')
elif button == 2:
    print('사이다')
elif button == 3:
    print('환타')
else:
    print('제공하지 않는 메뉴')
```

```
버튼을 눌러주세요: 3
환타
```

1. 조건문
- 중첩 조건문

- if 조건문 안에 if 조건문이 있는 것
- 논리연산자로 두 가지 이상 조건을 묶어 평가 가능

1. 나이와 점수를 입력 받는다
2. 합격조건을 중첩 조건문으로 나타낸다
- 나이가 20살 이상
- 점수는 80점 이상

```
age = int(input("나이를 입력하세요 : "))
score = int(input("점수를 입력하세요 : "))

if age >= 20 :
    if score >= 80 :
        print("합격입니다!")
    else :
        print("점수가 낮아 불합격입니다!")
else :
    print("너무 어려서 불합격입니다!")
```

```
나이를 입력하세요 : 22
점수를 입력하세요 : 90
합격입니다!
```

```
# 논리 연산자를 사용한 if 조건문
if age >= 20 and score >= 80 :
    print("합격입니다!")
else :
    print("불합격입니다!")
```

2. 반복문
- 반복문 이란 ? & 반복문이 필요한 이유

“ ‘Hello, bitamin!’ 을 100번 출력해보자 ! ”

Sol 1) 직접 print를 100번 한다 ^^

```
# print 100번 사용
print('Hello, bitamin!')
print('Hello, bitamin!')
print('Hello, bitamin!')
print('Hello, bitamin!')
# ... (생략)
print('Hello, bitamin!')
print('Hello, bitamin!')
print('Hello, bitamin!')
print('Hello, bitamin!')
```

Sol 2) 반복문을 구현한다

```
for i in range(100):
    print('Hello, bitamin!')
```

```
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
```

2. 반복문
- 반복문 이란 ? & 반복문이 필요한 이유

“ ‘Hello, bitamin!’ 을 100번 출력해보자 ! ”

Sol 1) 직접 print를 100번 한다 ^^

- 코드를 붙여 넣는데 시간이 오래 걸림
- 프로그래밍 측면에서 비효율적
- 반복되는 작업을 처리하는 반복문의 기능을 제공

Sol 2) 반복문을 구현한다

```
for i in range(100):
    print('Hello, bitamin!')
```

```
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
Hello, bitamin!
```

2. 반복문
- 반복문 이란 ? & 반복문이 필요한 이유

“ ‘Hello, bitamin!’ 을 100번 출력해보자 ! ”

Sol 1) 직접 print를 100번 한다 ^^

- 코드를 붙여 넣는데 시간이 오래 걸림
- 프로그래밍 측면에서 비효율적
- 반복되는 작업을 처리하는 반복문의 기능을 제공

Sol 2) 반복문을 구현한다

- 반복적인 작업을 처리하는 도구
- 코드 작업에서 가장 많이 사용하는 구문 중 하나

2. 반복문
- for와 range의 동작과정

변수,
범위 내 각각의 숫자가 여기에 저장됨

0~99까지의 숫자 범위,
반복 횟수

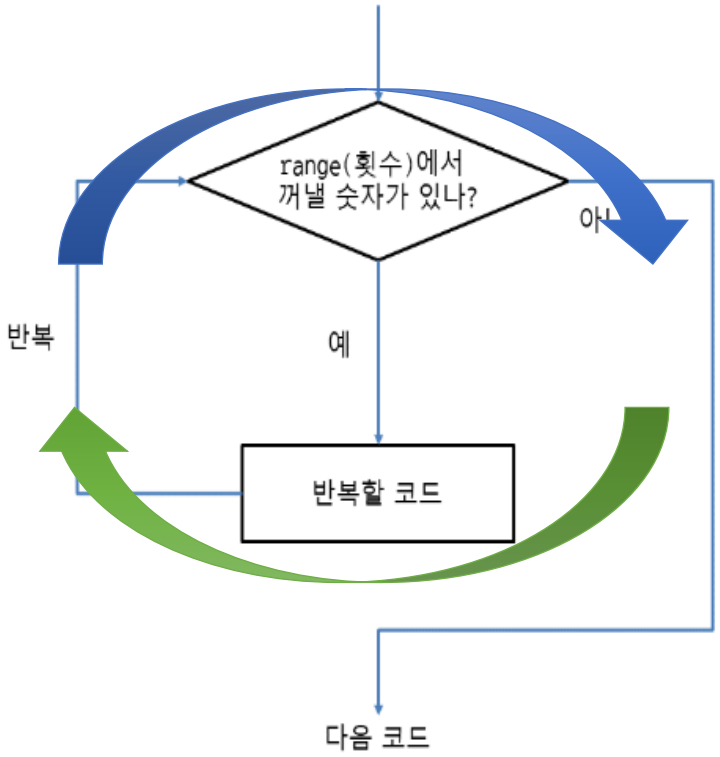
```
for i in range(100):
    print('Hello, bitamin!')
```

변수에 숫자가 저장될 때마다 수행할 코드

즉, for문은 반복 횟수가 정해져 있을 때 주로 사용한다 !!

2. 반복문
- for와 range의 동작과정

순환하는 loop




```
for i in range(10):
    print('Hello, bitamin!',i)
print('반복문 끝 !')
```

```
Hello, bitamin! 0
Hello, bitamin! 1
Hello, bitamin! 2
Hello, bitamin! 3
Hello, bitamin! 4
Hello, bitamin! 5
Hello, bitamin! 6
Hello, bitamin! 7
Hello, bitamin! 8
Hello, bitamin! 9
반복문 끝 !
```

```
for i in range(10):
    print(i)
    i=10
```

```
0
1
2
3
4
5
6
7
8
9
```

출처 : 코딩도장

2. 반복문
- 다양한 for문의 사용

- range 증가 폭 사용하기

```
for i in range(10,0,-1):
    print(i)
```

```
10
9
8
7
6
5
4
3
2
1
```

```
for i in range(0,30,3):
    print(i)
```

```
0
3
6
9
12
15
18
21
24
27
```

- list 원소 순회하기

```
name = ['이소연', '김도경', '김지윤', '서현재']
for n in name:
    print(n)
```

```
이소연
김도경
김지윤
서현재
```

- tuple 원소 순회하기

```
name = ('이소연', '김도경', '김지윤', '서현재')
for n in name:
    print(n)
```

```
이소연
김도경
김지윤
서현재
```

2. 반복문
- 다양한 for문의 사용

- 문자열 순회하기

```
for i in 'hello bitamin!':
    print(i)
```

```
h
e
l
l
o

b
i
t
a
m
i
n
!
```

- dictionary 순회하기

```
bitamin = {'교육부':'김도경','기획부':'김지윤','총무부':'서현재'}
for depart in bitamin:
    print(depart)
```

key값만 출력됨
bitamin.keys()와 결과가 같다

```
교육부
기획부
총무부
```

```
bitamin = {'교육부':'김도경','기획부':'김지윤','총무부':'서현재'}
for name in bitamin.values():
    print(name)
```

```
김도경
김지윤
서현재
```

```
bitamin = {'교육부':'김도경','기획부':'김지윤','총무부':'서현재'}
for depart,name in bitamin.items():
    print(depart, name)
```

```
교육부 김도경
기획부 김지윤
총무부 서현재
```

2. 반복문
- for문 응용

enumerate 함수

- 순서와 리스트의 값을 전달하는 기능
- “열거하다”라는 뜻
- 순서가 있는 자료형 (list, set, tuple, dictionary, string)을 입력으로 받아 index 값을 포함하는 enumerate 객체를 리턴
- for문과 함께 자주 사용

```
name = ['이소연', '김도경', '김지윤', '서현재']
list(enumerate(name))
```

```
[(0, '이소연'), (1, '김도경'), (2, '김지윤'), (3, '서현재')]
```

```
name = ['이소연', '김도경', '김지윤', '서현재']
for index, name in enumerate(name):
    if index > 1:
        print(index, name)
```

```
2 김지윤
3 서현재
```

2. 반복문
- for문 응용

" 학생(1~5번)의 시험점수가 50,70,80,20,90점일 때, 시험 점수가 60점이 넘으면 합격, 그렇지 않으면 불합격이다. 각 학생이 합격인지 불합격인지 결과를 보여주는 코드를 구현해 보자 !!! "

Step1) 시험점수를 list로 만들기

Step2) 시험점수와 학생을 mapping하기

Step3) 주어진 조건을 if문을 활용하여 구현하기

Step4) for문을 활용하여 mapping된 자료형의 요소를 순회하며 주어진 조건에 따라 학생의 합불 여부를 결정하기

```
score = [50,70,80,20,90]
data = enumerate(score)

for idx,sc in data:
    if sc > 60:
        print(idx+1, ':', sc, '(합격)')
    else:
        print(idx+1, ':', sc, '(불합격)')
```

```
1 : 50 (불합격)
2 : 70 (합격)
3 : 80 (합격)
4 : 20 (불합격)
5 : 90 (합격)
```

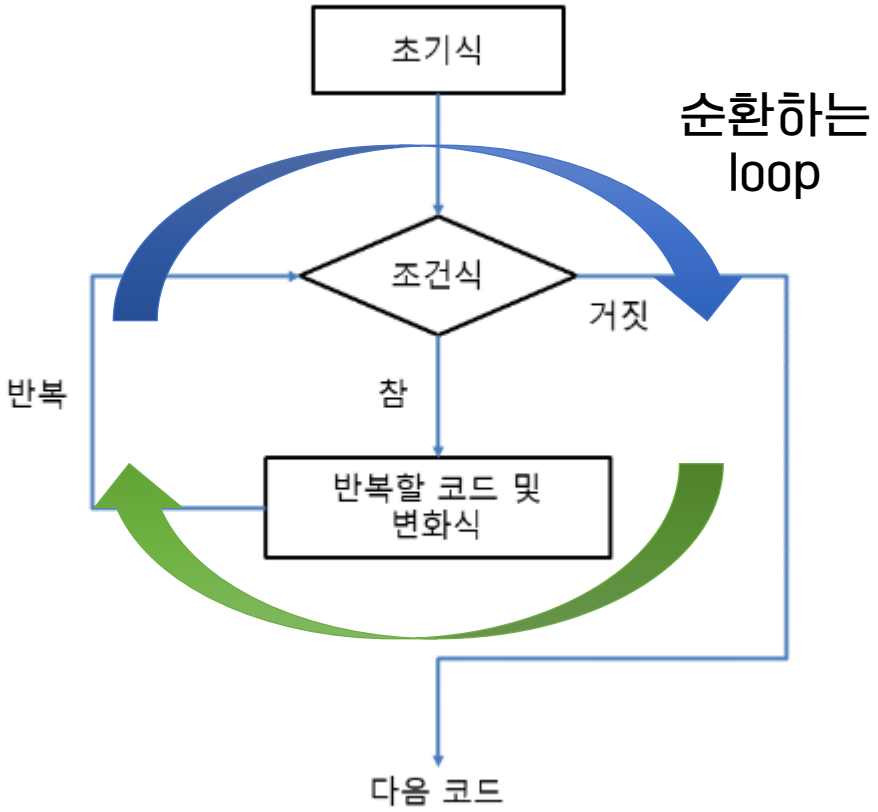
2. 반복문
- while문의 기본구조 & 동작과정

기본구조

```
while 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
    ...
```

- while 뒤의 조건문이 참인 동안 반복 수행
- 조건문이 False가 되면 멈추고 다음 코드를 실행
- 반드시 반복을 멈추게 하는 요소가 필요
- while문에 속하는 문장은 들여쓰기 필수

동작과정



출처: 코딩도장

2. 반복문
- while문의 기본구조 & 동작과정

```
i=0                    초기식
while i < 100:         조건식, True일 때만 동작
    print("Hello, bitamin!")   반복할 코드
    i += 1             변화식
```

즉, while문은 반복 횟수가 정해져 있지 않을 때 주로 사용한다 !!

2. 반복문
- 다양한 while문의 활용

“ ‘Hello, bitamin!’ 을 100번 출력해보자 ! ”

```
i=0
while i < 100:
    print('Hello, bitamin!',i)
    i += 1
```

```
Hello, bitamin! 0
Hello, bitamin! 1
Hello, bitamin! 2
Hello, bitamin! 3
Hello, bitamin! 4
Hello, bitamin! 5
Hello, bitamin! 6
Hello, bitamin! 7

Hello, bitamin! 94
Hello, bitamin! 95
Hello, bitamin! 96
Hello, bitamin! 97
Hello, bitamin! 98
Hello, bitamin! 99
```

“ 열 번 찍어 안 넘어가는 나무 없다 ”

```
treeHit = 0
while treeHit < 10:
    treeHit = treeHit + 1
    print("나무를",treeHit,"번 찍었습니다")
    if treeHit == 10:
        print("나무 넘어갑니다")
```

```
나무를 1 번 찍었습니다
나무를 2 번 찍었습니다
나무를 3 번 찍었습니다
나무를 4 번 찍었습니다
나무를 5 번 찍었습니다
나무를 6 번 찍었습니다
나무를 7 번 찍었습니다
나무를 8 번 찍었습니다
나무를 9 번 찍었습니다
나무를 10 번 찍었습니다
나무 넘어갑니다
```

출처: 점프 투 파이썬

2. 반복문
- 무한 loop

" while문에 반복을 멈추는 요소가 없으면 무한 loop를 돌게 된다 "

BITAmin

- while에 True를 지정할 때
- 0이 아닌 숫자는 True로 취급
- 내용이 있는 문자열은 True로 취급

```
In [*]: while True:
            print("Hello~")
```

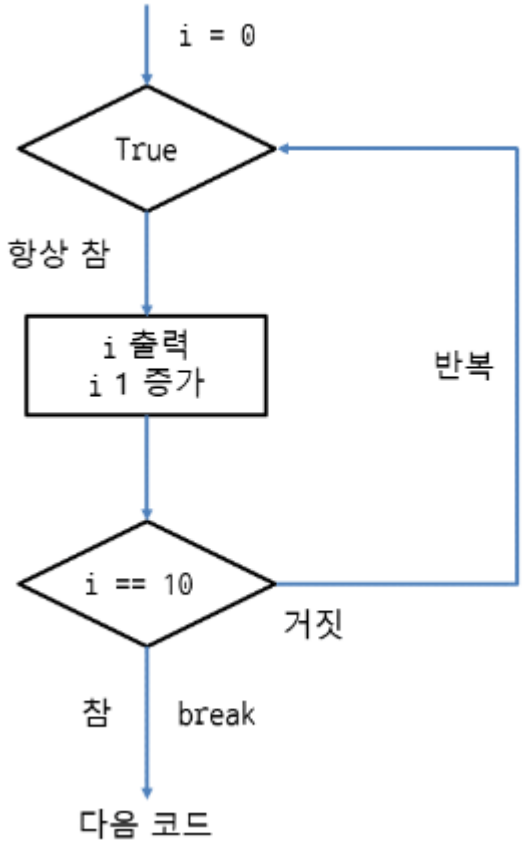
```
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
Hello~
```

2. 반복문
- 반복문 강제로 빠져나가기 (break문)

- while문

```
i=0
while True:
    print(i)
    i += 1
    if i==10:
        break
```

```
0
1
2
3
4
5
6
7
8
9
```



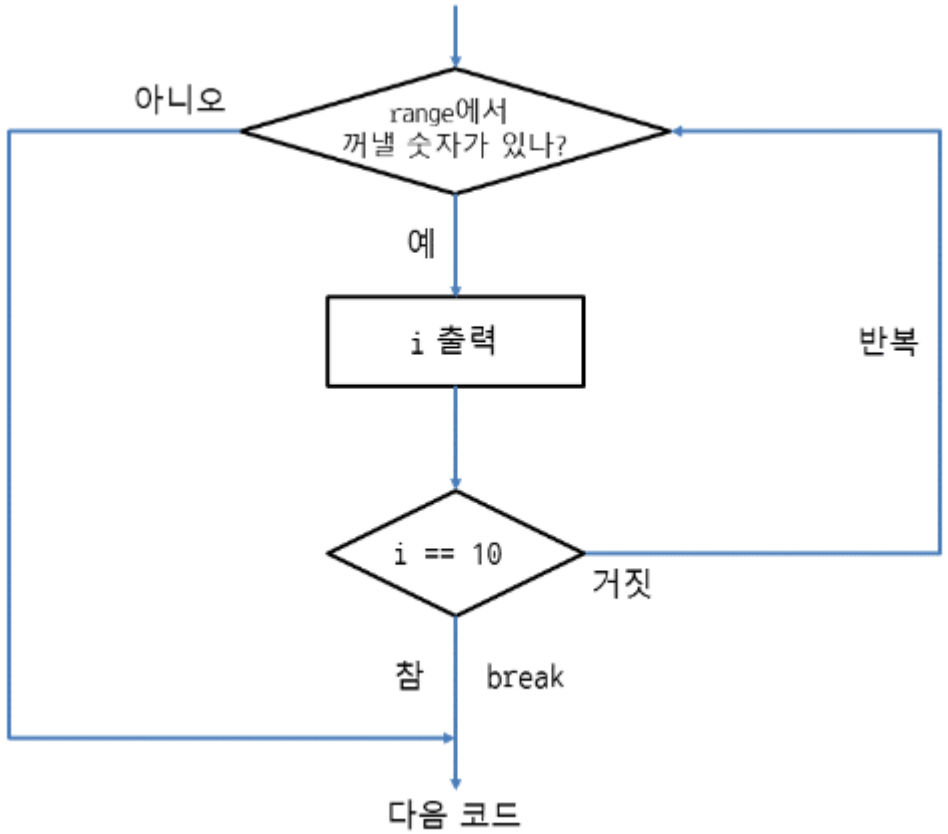
출처: 코딩도장

2. 반복문
- 반복문 강제로 빠져나가기 (break문)

- for문

```
for i in range(10000000000):
    print(i)
    if i==10:
        break
```

```
0
1
2
3
4
5
6
7
8
9
10
```



출처: 코딩도장

2. 반복문
- 반복문 강제로 빠져나가기 (break문)

“ 자판기에 한 잔에 300원인 커피가 10잔이 남아있다. 커피가 다 떨어질 때까지의 과정을 while문을 사용하여 구현해 보자 ”

Step1) 커피는 10잔이 남아있음을 변수로 저장

Step2) 돈을 받을 때 마다 커피가 한 잔 씩 소진

Step3) 커피가 모두 소진되면 판매 중지

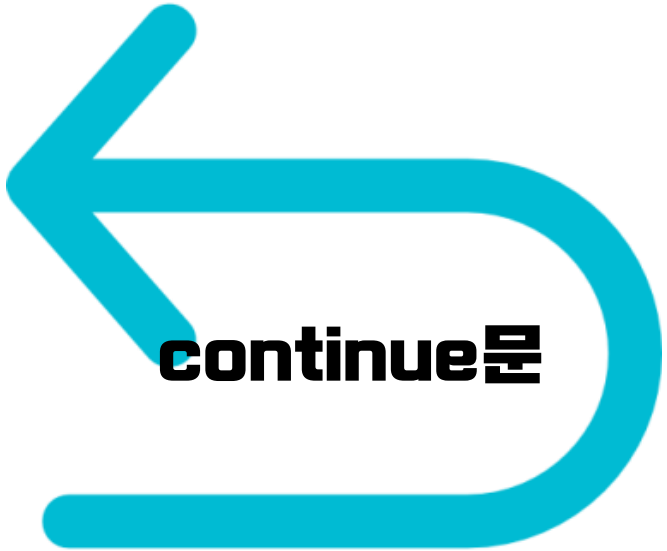
```
money = 300
coffee = 10

while money:
    print("돈을 받았으니 커피를 드립니다.")
    coffee = coffee - 1
    print("남은 커피는", coffee, "잔 입니다.")
    if coffee == 0:
        print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
        break
```

```
돈을 받았으니 커피를 드립니다.
남은 커피는 9 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 8 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 7 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 6 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 5 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 4 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 3 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 2 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 1 잔 입니다.
돈을 받았으니 커피를 드립니다.
남은 커피는 0 잔 입니다.
커피가 다 떨어졌습니다. 판매를 중지합니다.
```

2. 반복문
- 처음으로 돌아가기 (continue문)

- 반복문을 전체를 빠져나가지 않고 반복문의
  맨 처음으로 돌아가고 싶을 땐?
- 특정 조건에서 코드 실행을 건너뛰고 싶을 땐?



2. 반복문
- 처음으로 돌아가기 (continue문)

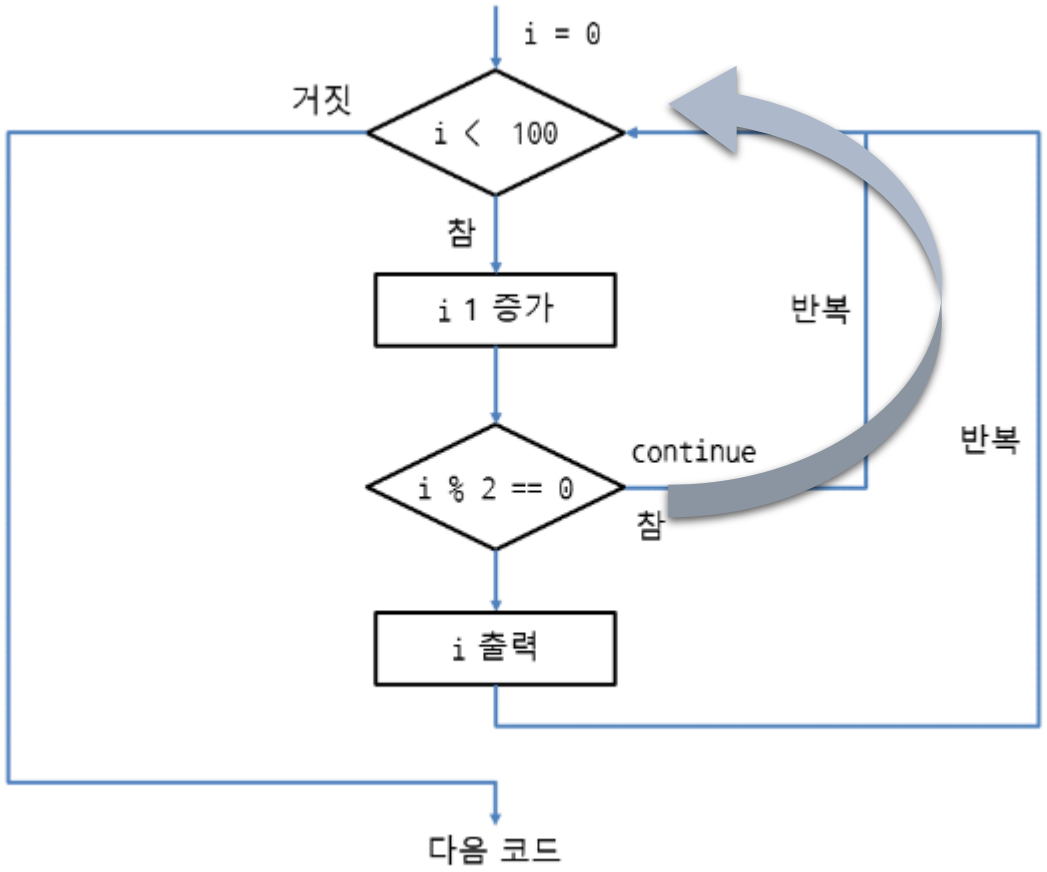
- while문

```
i=0
while i<100:
    i+=1
    if i%2 == 0:
        continue
    print(i)
```

i가 짝수일 때 print하지 않고, while문의 조건을 판단하는 곳으로 돌아감

```
1
3
5
7
9
11
13
15
17

87
89
91
93
95
97
99
```



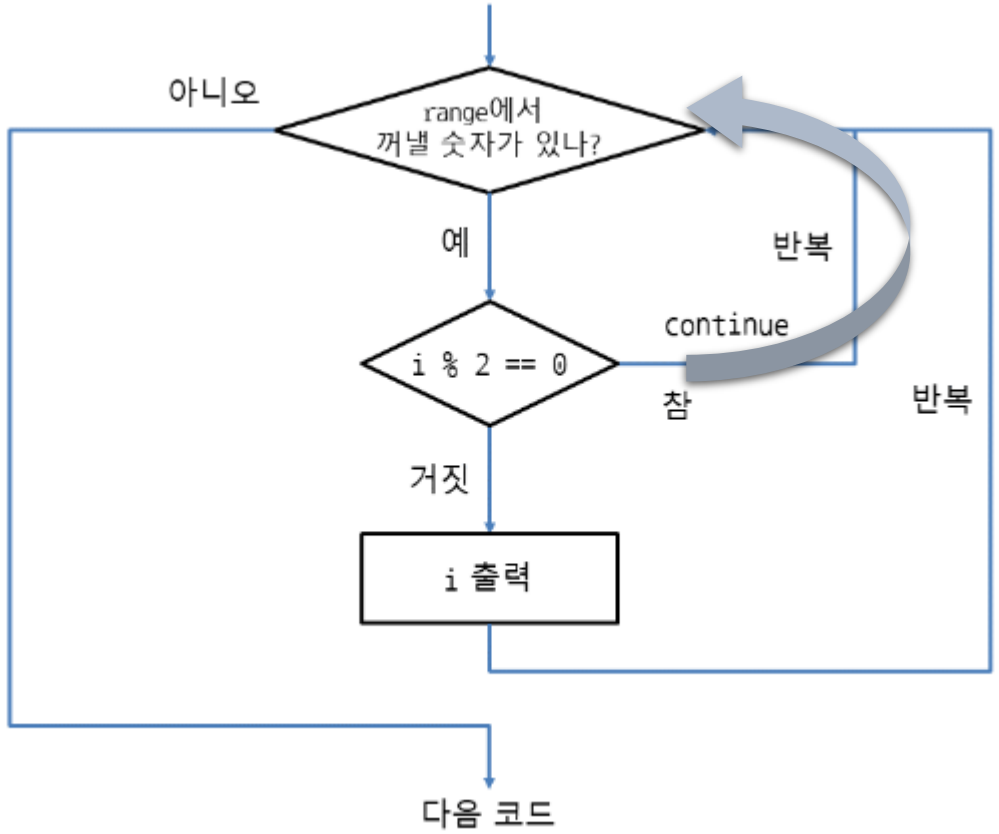
2. 반복문
- 처음으로 돌아가기 (continue문)

- for문

```
for i in range(100):
    if i%2 == 0:
        continue
    print(i)
```

i가 짝수일 때 print하지 않고, for문의 변수 지정하는 곳으로 돌아감

```
1
3
5
7
9
11
13
15
17
89
91
93
95
97
99
```



2. 반복문
- 실행할 코드가 없을 때 (pass문)

- 아무 일도 하지 않고 그냥 넘어간다는 뜻
- if 다음에 아무 코드도 넣지 않으면 오류가 발생하므로 조건문의 형태를 유지하기 위해 사용
- 나중에 작성해야 할 코드를 표시할 때 사용

```
for i in range(100):
    if i%2 == 0: i가 짝수일 때 실행할 코드가 없음
        pass
    print(i)
```

```
0
1
2
3
4
5
6
7
8
9


95
96
97
98
99
```

2. 반복문
- 중첩 루프

- 반복문 여러 개가 겹쳐 있는 구조
- 2차원 데이터를 다룰 때 자주 사용

“ 2차원 리스트를 1차원으로 바꿔보기 ”

```
matrix = [[1,2,3],[4,5,6],[7,8,9]]
new = []

for i in matrix:
    for j in i:
        print(j)
        new.append(j)
new
```

```
1
2
3
4
5
6
7
8
9

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

2. 반복문
- 중첩 루프

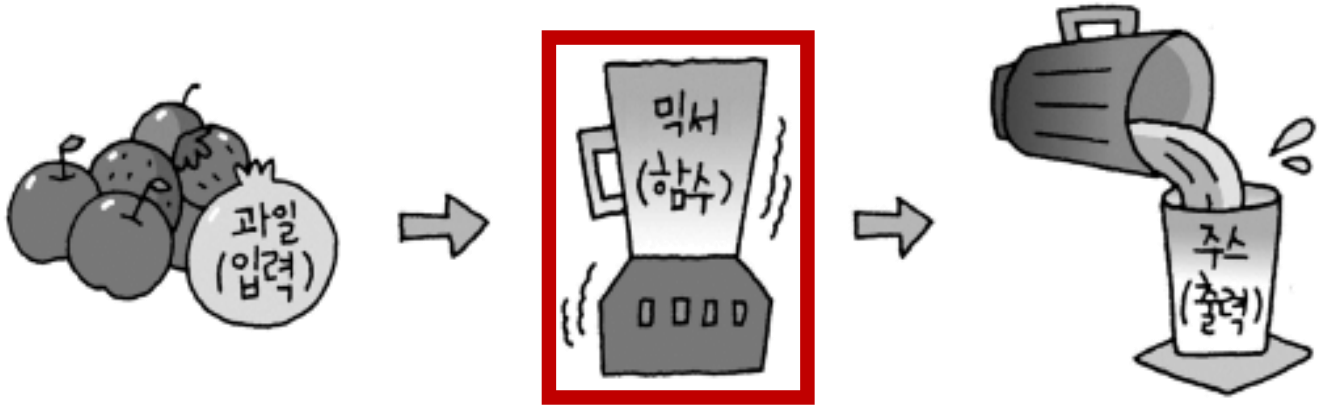
- 반복문 여러 개가 겹쳐 있는 구조
- 2차원 데이터를 다룰 때 자주 사용

“ 10 x 10 구구단 만들어 보기 ”

```
for i in range(1,11):
    for j in range(1,11):
        print(i,'*',j,'=',i*j)
    print('----------------------------------')
```

```
1 * 1 = 1
1 * 2 = 2
1 * 3 = 3
1 * 4 = 4
1 * 5 = 5
1 * 6 = 6
1 * 7 = 7
1 * 8 = 8
1 * 9 = 9
1 * 10 = 10
----------------------------------
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
2 * 10 = 20
----------------------------------
----------------------------------
9 * 1 = 9
9 * 2 = 18
9 * 3 = 27
9 * 4 = 36
9 * 5 = 45
9 * 6 = 54
9 * 7 = 63
9 * 8 = 72
9 * 9 = 81
9 * 10 = 90
----------------------------------
10 * 1 = 10
10 * 2 = 20
10 * 3 = 30
10 * 4 = 40
10 * 5 = 50
10 * 6 = 60
10 * 7 = 70
10 * 8 = 80
10 * 9 = 90
10 * 10 = 100
----------------------------------
```

2. 함수 (function)
- 함수란?



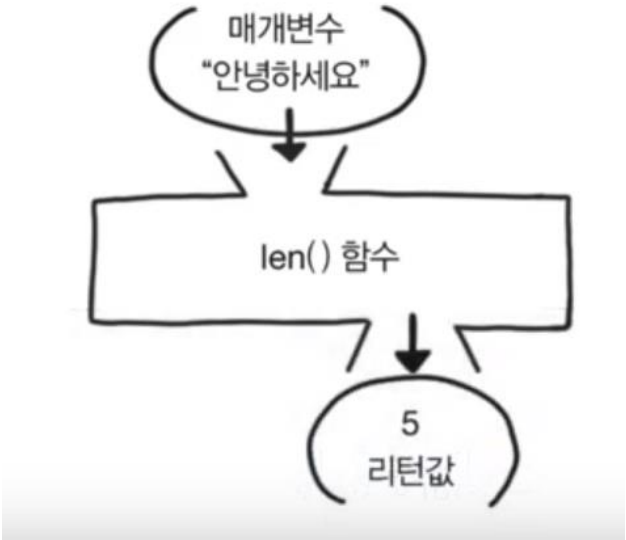
- 과일을 입력 받아 주스를 출력하는 함수
- 입력값을 가지고 어떤 일을 수행한 다음
  그 결과물을 내어 놓는 것

2. 함수 (function)
- 함수를 사용하는 이유

단순히 복사 후 붙여넣기
- 코드를 유지 보수하기 어려워짐

함수(function)라는 형태로 코드를 작성하자!
- 필요할 때 마다 '호출'하여 사용

2. 함수 (function)
- 내장함수



- Python에서 제공되는 기본함수
- import가 필요 없기 때문에 설정 없이 바로 사용 가능

		내장 함수		
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

출처: https://docs.python.org/ko/3/library/functions.html

2. 함수 (function)
- 함수 정의하기

기본구조

```
def 함수이름(매개변수):
    <수행할 문장>
    ...
    return 결과값
```

- 사용할 기능을 만드는 것
- 키워드 : def (definition의 줄임말)
- 함수이름, 실행 구문, 매개변수, 결과값으로 구성

매개변수와 인수

```
def add(a, b): 매개변수
    return a+b

print(add(3, 4)) 인자
```

```
7
```

헷갈리지 말아욥 ~!

- 매개변수(parameter) : 함수에 입력으로 전달된 값을 받는 변수
- 인자(arguments) : 함수를 호출할 때 전달하는 입력값

2. 함수 (function)
- 다양한 함수 형태

입력값이 없는 함수

```
def say():
    return 'Hi'
```

```
a = say()
print(a)
```

```
Hi
```

- 괄호 안에 아무 값도 넣지 않아야 함
- 입력값은 없지만 Hi라는 문자열을 반환
- a 라는 변수에 Hi라는 문자열이 대입됨

결과값이 없는 함수

```
def add(a,b):
    print(a,'와',b,'의 합은',a+b,'입니다.')
```

```
a = add(3,4)
```

```
3 와 4 의 합은 7 입니다.
```

```
print(a)
```

```
None
```

- 문장은 출력되지만 단지 <수행할 문장>의 일부일 뿐 결과값은 없는 것 !!
- 결과값은 오직 return 명령어로만

2. 함수 (function)
- 다양한 함수 형태

여러 개의 입력값을 받는 함수

```
def add_all(*args):
    result = 0

    for i in args:
        result = result + i

    return result
```

```
add_all(1,2,3,4,5,6,7,8,9,10)
```

```
55
```

- 매개변수 앞에 '*' 을 붙이면 입력값을 전부 모아서 tuple로 만든다

혼용도 가능 !

```
def add_mul(choice,*args):
    if choice == "add":
        result=0
        for i in args:
            result=result+i

    elif choice == "mul":
        result=1
        for i in args:
            result=result*i

    return result
```

```
add_mul('add', 1,2,3,4,5)
```

```
15
```

```
add_mul('mul', 1,2,3,4,5)
```

```
120
```

2. 함수 (function)
- 다양한 함수 형태

여러 개의 결과값을 반환하는 함수 ?

```
def add_and_mul(a,b):
    return a+b, a*b
```

```
result = add_and_mul(3,4)
```

- 결과값은 두개 but 변수는 하나
- 오류가 발생하지 않을까???

2. 함수 (function)
- 다양한 함수 형태

여러 개의 결과값을 반환하는 함수 ?

```
def add_and_mul(a,b):
    return a+b, a*b
```

```
result = add_and_mul(3,4)
```

- 결과값은 두개 but 변수는 하나
- 오류가 발생하지 않을까???

NOPE !!!!!!

```
result
```

```
(7, 12)
```

- 함수의 결과값은 항상 1개
- 두 결과값이 하나의 tuple로 묶여 반환

2개의 결과값처럼 받고 싶다면 ??

```
result1, result2 = add_and_mul(3,4)
```

```
result1
```

```
7
```

```
result2
```

```
12
```

2. 함수 (function)
- 다양한 함수 형태

매개변수에 default값 설정하기

```
def hello(name, age, univ='국민대학교'):
    print("안녕하세요")
    print("저는",name,"입니다")
    print("나이는",age,"살 입니다")
    if univ=='국민대':
        print("국민대학교 학생 입니다~!")
    else:
        print(univ,"학생 입니다~!")
```

```
hello('이소연',22)
```

```
안녕하세요
저는 이소연 입니다
나이는 22 살 입니다
국민대학교 학생 입니다~!
```

```
hello('이소연',22, '서울대학교')
```

```
안녕하세요
저는 이소연 입니다
나이는 22 살 입니다
서울대학교 학생 입니다~!
```

- univ 매개변수의 default값을 '국민대학교'로 설정
- 매개변수에 들어갈 값이 항상 변하는 것이 아닐 경우 기본값 설정해 두기 !!
- default 값을 설정한 매개변수는 항상 뒤쪽에 놓아야 함 !!!

```
def hello(name, univ='국민대학교', age):
```

```
  File "<ipython-input-113-4f93596f077a>", line 1
    def hello(name, univ='국민대학교', age):
                                      ^
SyntaxError: non-default argument follows default argument
```

2. 함수 (function)
– 익명 함수 lambda

```
lambda 매개변수: 식
```

- 이름이 없는 함수
- 필요한 곳에서 즉시 사용하고 버리는 일시적인 함수
- 식 형태로 되어 있어 람다 표현식(lambda expression)이라고 부름
- 간편하게 작성 가능하여 다른 함수의 인수로 넣을 때 주로 사용

호출방법 1) 변수에 할당하여 호출

```
lambda x,y: x + y
```

```
<function __main__.<lambda>(x, y)>
```

```
add = lambda x,y: x + y
add(2,3)
```

```
5
```

호출방법 2) 람다 표현식 자체를 호출

```
(lambda x,y: x + y)(2,3)
```

```
5
```

2. 함수 (function)
– 익명 함수 lambda

람다 표현식을 함수의 인수로 사용하기

```
map(lambda x: x + 10, range(10))
```

```
<map at 0x1e5e10a9a88>
```

```
list(map(lambda x: x + 10, range(10)))
```

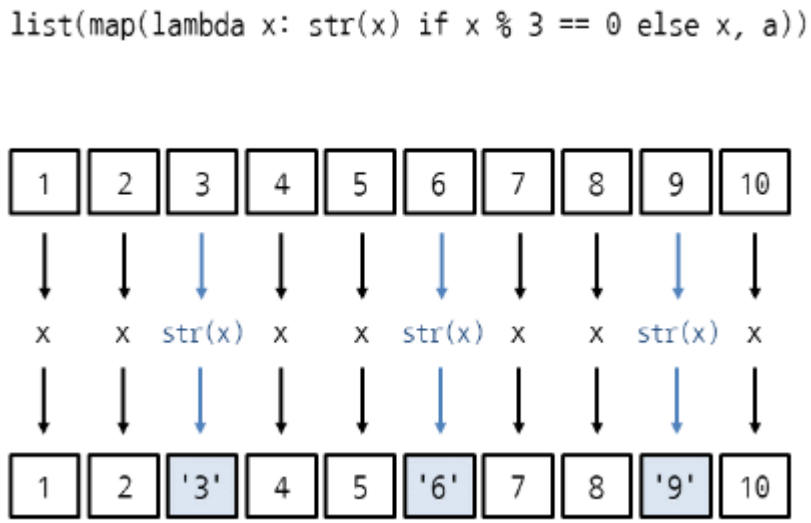
```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

map()
- 리스트의 요소를 지정된 함수로 처리해주는 함수
- 여러 개의 데이터를 한 번에 다른 형태로 변환하기 위해 사용

람다 표현식에 조건부 표현식 사용하기

```
list(map(lambda x: str(x) if x % 3 == 0 else x, range(1,11)))
```

```
[1, 2, '3', 4, 5, '6', 7, 8, '9', 10]
```



2. 함수 (function)
- 익명 함수 lambda

람다 표현식을 함수의 인수로 사용하기

```
map(lambda x: x + 10, range(10))
```

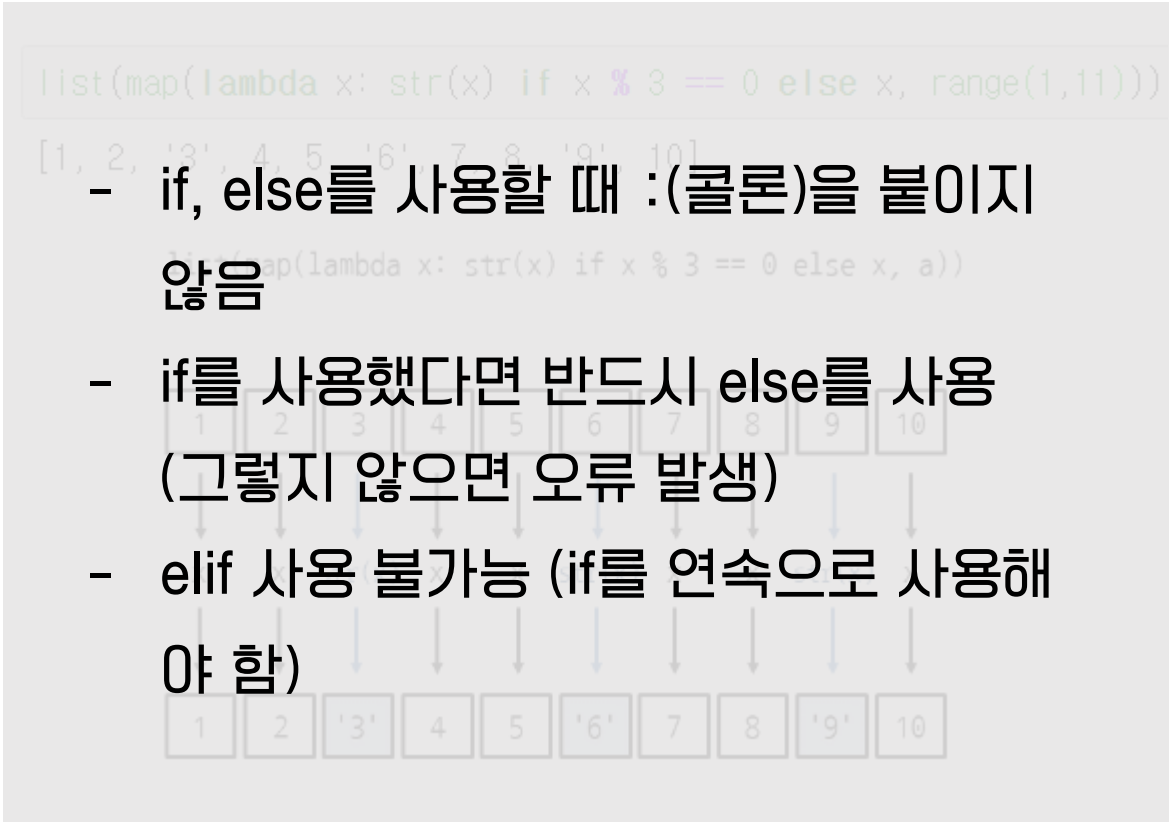
```
<map at 0x1e5e10a9a88>
```

```
list(map(lambda x: x + 10, range(10)))
```

```
[10, 11, 12, 13, 14, 15, 16, 17, 18, 19]
```

map()
- 리스트의 요소를 지정된 함수로 처리해주는 함수
- 여러 개의 데이터를 한 번에 다른 형태로 변환하기 위해 사용

람다 표현식에 조건부 표현식 사용하기



- if, else를 사용할 때 :(콜론)을 붙이지 않음
- if를 사용했다면 반드시 else를 사용 (그렇지 않으면 오류 발생)
- elif 사용 불가능 (if를 연속으로 사용해야 함)

3. 정규 표현식
- 개념 및 필요한 이유

정규 표현식 (Regular Expression) 이란?

- 문자열을 처리할 때 사용하는 기법
- Python만의 고유 문법이 아님
- 문자열을 처리하는 모든 곳에서 사용
- 모든 문자 하나는 일반 문자열 하나와 매칭
- '정규식' 이라고도 함

정규 표현식 (Regular Expression)이 필요한 이유

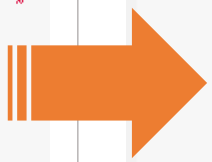
- 생산성과 효율성의 문제 !!
- 문자열의 검색 / 추출 / 치환을 위한 용도
- 일반적인 조건문에 비해 코드가 훨씬 간단
- 그만큼 가독성이 떨어짐

3. 정규 표현식
- 개념 및 필요한 이유

```
//text에서 어떤 tag가 있는지 모조리 뽑아내서 list에 담아 반환
private List<String> tagPicker(String text) {

    List<String> tagList = new ArrayList<String>();
    int textLength = text.length();

    for(int i=0; i<textLength; i++){
        StringBuilder sb = new StringBuilder();
            if(text.charAt(i)=='#'){
                for(int j=i+1; j<textLength; j++){
                    if(text.charAt(j)=='₩n' || text.charAt(j)==' ' || text.charAt(j)=='#' || text.charAt(j)=='%'
                        || text.charAt(j)=='.' || text.charAt(j)==',' || j==(textLength-1) ){
                        if(sb.length()!=0){
                            if(j==(textLength-1))
                                sb.append(text.charAt(j));
                                tagList.add(sb.toString());
                            }
                            i=j-1;
                            break;
                        } else {
                            sb.append(text.charAt(j));
                        }
                        LOGGER.debug("TAG : " + sb.toString());
                    }
                }
            }
    return tagList;
}
```



```
//text에서 어떤 tag가 있는지 모조리 뽑아내서 list에 담아 반환
private List<String> getTagListFromText(String text) {

    List<String> tagList = new ArrayList<String>();

    Pattern p = Pattern.compile("#([a-zA-Z0-9가-힣]+)");
    Matcher mc = p.matcher(text);

    while(mc.find()){
        tagList.add(mc.group(1));
    }
    return tagList;
}
```

3. 정규 표현식
- re 모듈

```
import re
```

- 정규 표현식을 지원하기 위해 re (regular expression의 약어) 모듈을 제공
- Python 설치할 때 자동으로 설치되는 기본 라이브러리
- 11개의 메타문자(meta characters) 존재

* 메타문자(meta characters) 란?

```
. ^ $ * + ? { } [ ] \ | ( )
```

- 특수한 기능을 하는 문자
- 각각의 문자 하나에 매칭되지 않음

ex) 일반 문자 a는 문자 'a'에 매칭 but 소괄호 (는 문자 '('와 매칭되지 않음

- 메타문자 앞에 \ 를 붙여주면 일반 문자처럼 한 글자에 매칭됨

ex) \(는 '\('와 매칭됨

3. 정규 표현식
-정규 표현식 용어들

표현식	의미
^x	문자열의 시작을 표현하며 x 문자로 시작됨을 의미한다.
x$	문자열의 종료를 표현하며 x 문자로 종료됨을 의미한다.
.x	임의의 한 문자의 자리수를 표현하며 문자열이 x 로 끝난다는 것을 의미한다.
x+	반복을 표현하며 x 문자가 한번 이상 반복됨을 의미한다.
x?	존재여부를 표현하며 x 문자가 존재할 수도, 존재하지 않을 수도 있음을 의미한다.
x*	반복여부를 표현하며 x 문자가 0번 또는 그 이상 반복됨을 의미한다.
x|y	or 를 표현하며 x 또는 y 문자가 존재함을 의미한다.
(x)	그룹을 표현하며 x 를 그룹으로 처리함을 의미한다.
(x)(y)	그룹들의 집합을 표현하며 앞에서 부터 순서대로 번호를 부여하여 관리하고 x, y 는 각 그룹의 데이터로 관리된다.
(x)(?:y)	그룹들의 집합에 대한 예외를 표현하며 그룹 집합으로 관리되지 않음을 의미한다.
x{n}	반복을 표현하며 x 문자가 n번 반복됨을 의미한다.
x{n,}	반복을 표현하며 x 문자가 n번 이상 반복됨을 의미한다.
x{n,m}	반복을 표현하며 x 문자가 최소 n번 이상 최대 m 번 이하로 반복됨을 의미한다.

표현식	의미
[xy]	문자 선택을 표현하며 x 와 y 중에 하나를 의미한다.
[^xy]	not 을 표현하며 x 및 y 를 제외한 문자를 의미한다.
[x-z]	range를 표현하며 x ~ z 사이의 문자를 의미한다.
₩^	escape 를 표현하며 ^ 를 문자로 사용함을 의미한다.
₩b	word boundary를 표현하며 문자와 공백사이의 문자를 의미한다.
₩B	non word boundary를 표현하며 문자와 공백사이가 아닌 문자를 의미한다.
₩d	digit 를 표현하며 숫자를 의미한다.
₩D	non digit 를 표현하며 숫자가 아닌 것을 의미한다.
₩s	space 를 표현하며 공백 문자를 의미한다.
₩S	non space를 표현하며 공백 문자가 아닌 것을 의미한다.
₩t	tab 을 표현하며 탭 문자를 의미한다.
₩v	vertical tab을 표현하며 수직 탭(?) 문자를 의미한다.
₩w	word 를 표현하며 알파벳 + 숫자 + _ 중의 한 문자임을 의미한다.
₩W	non word를 표현하며 알파벳 + 숫자 + _ 가 아닌 문자를 의미한다.

출처 : https://hamait.tistory.com/342

3. 정규 표현식
-정규 표현식 용어들

표현식	의미
^x	문자열의 시작을 표현하며 x 문자로 시작됨을 의미한다.
x$	문자열의 종료를 표현하며 x 문자로
.x	임의의 한 문자의 자리수를 표현하
x+	반복을 표현하며 x 문자가 한번 이
x?	존재여부를 표현하며 x 문자가 존
x*	반복여부를 표현하며 x 문자가 0번
x|y	or 를 표현하며 x 또는 y 문자가 존
(x)	그룹을 표현하며 x 를 그룹으로 처
(x)(y)	그룹들의 집합을 표현하며 앞에서 그룹의 데이터로 관리된다.
(x)(?:y)	그룹들의 집합에 대한 예외를 표현
x{n}	반복을 표현하며 x 문자가 n번 반
x{n,}	반복을 표현하며 x 문자가 n번 이상 반복됨을 의미한다.
x{n,m}	반복을 표현하며 x 문자가 최소 n번 이상 최대 m 번 이하로 반복됨을 의미한다.

표현식	의미
[xy]	문자 선택을 표현하며 x 와 y 중에 하나를 의미한다.
	를 의미한다.
	의미한다.
	함을 의미한다.
	백사이의 문자를 의미한다.
	와 공백사이가 아닌 문자를 의미한다.
	을 의미한다.
	다.
	닌 것을 의미한다.
	자를 의미한다.
₩w	word 를 표현하며 알파벳 + 숫자 + _ 중의 한 문자임을 의미한다.
₩W	non word를 표현하며 알파벳 + 숫자 + _ 가 아닌 문자를 의미한다.

BITAmin

꿀 Tip !!

1. 대문자와 소문자는 서로 반대 역할을 한다
2. ^문자열 은 처음과 일치함을 의미하지만,

   [^문자열]에서는 not을 의미한다

출처 : https://hamait.tistory.com/342

3. 정규 표현식
- re 모듈 (findall)

```
import re
```

```
s = 'Hello, BITAmin!'
```

- 문자열 중 패턴과 일치되는 모든 부분을 찾는다

1. [ ] : 문자 클래스, 어떤 것이든 들어갈 수 있다

```
re.findall(r'o',s)
```

```
['o']
```

```
re.findall(r'o|e', s) # 'o'또는'e' 찾기
```

```
['e', 'o']
```

```
re.findall(r'a|e|i|o|u', s)
```

```
['e', 'o', 'i']
```

```
re.findall(r'[aeiou]', s)
```

```
['e', 'o', 'i']
```

```
re.findall(r'[a-z]', s) # 소문자 a부터 z까지
```

```
['e', 'l', 'l', 'o', 'm', 'i', 'n']
```

```
re.findall(r'[A-Z]', s) # 대문자 A부터 Z까지
```

```
['H', 'B', 'I', 'T', 'A']
```

```
re.findall(r'[A-Za-z]', s) # 소문자 대문자 모두
```

```
['H', 'e', 'l', 'l', 'o', 'B', 'I', 'T', 'A', 'm', 'i', 'n']
```

```
re.findall(r'[^aeiou]', s) # 모음을 제외하고
```

```
['H', 'l', 'l', ',', ' ', 'B', 'I', 'T', 'A', 'm', 'n', '!']
```

3. 정규 표현식
- re 모듈 (findall)

2. ₩d : 숫자 0-9 / ₩D : 숫자가 아닌 것

```
re.findall(r'₩d', 'abcd1234')  ## ₩d는 0에서 9까지
```

```
['1', '2', '3', '4']
```

```
re.findall(r'[0-9]', 'abcd1234')
```

```
['1', '2', '3', '4']
```

```
re.findall(r'₩D', 'abcd1234')  ## ₩D는 숫자가 아닌것
```

```
['a', 'b', 'c', 'd']
```

```
re.findall(r'[^0-9]', 'abcd1234')
```

```
['a', 'b', 'c', 'd']
```

3. ₩s: 공백 / ₩S: 공백이 아닌 것

```
re.findall(r'₩s', 'abcd 1234')  ## ₩s는 공백
```

```
[' ']
```

```
re.findall(r'₩S', 'abcd 1234')  ## ₩S는 공백이 아닌것
```

```
['a', 'b', 'c', 'd', '1', '2', '3', '4']
```

4. ₩w: word, 일반적인 문자 (알파벳, 숫자, _)
₩W: non-word 일반적인 문자가 아닌 것

```
re.findall(r'₩w', '가나다ab_cd 1234') ## ₩w는 일반적인 문자
```

```
['가', '나', '다', 'a', 'b', '_', 'c', 'd', '1', '2', '3', '4']
```

```
re.findall(r'₩W', '가나다ab_cd 1234')
```

```
[' ']
```

3. 정규 표현식
- re 모듈 (sub)

- 문자열 중 패턴과 일치되는 모든 부분을 대체한다

1. 특정 문자열 대체

```
re.sub(r'Hello', 'Hi', s)
```

```
'Hi, BITAmin!'
```

```
re.sub(r'll', 'ㄹㄹ', s)
```

```
'Heㄹㄹo, BITAmin!'
```

2. 범위 지정

```
re.sub(r'l{2}', 'ㄹㄹ', s)
```

```
'Heㄹㄹo, BITAmin!'
```

```
re.sub(r'[^aeiou]{2,5}', '##', s)
```

```
'He##o####i##'
```

```
re.sub(r'[^aeiou]{,2}', '##', s)
```

```
'####e####o##########i####'
```

```
re.sub(r'₩w+', '##', s)
```

```
'##, ##!'
```

```
re.sub(r'₩W+', '##', s)
```

```
'Hello##BITAmin##'
```

```
re.sub('http?://', '', 'http://google.com')
```

```
'google.com'
```

```
re.sub('http{0,1}://', '', 'http://google.com')
```

```
'google.com'
```

3. 정규 표현식
- re 모듈 (match와 search)

- match : 문자열의 시작부터 비교하여 해당 패턴이 있는지 확인한다

```
re.match('a','aba')
```

```
<re.Match object; span=(0, 1), match='a'>
```

```
re.match('a','baa')
```

```
print(re.match('a','baa'))
```

```
None
```

- search : 문자열 전체를 검색하여 해당 패턴이 있는지 확인한다

```
re.search('a','aba')
```

```
<re.Match object; span=(0, 1), match='a'>
```

```
re.search('a','baa')
```

```
<re.Match object; span=(1, 2), match='a'>
```

공통점 :해당 패턴이 발견되면 검색은 멈춘다 !!
cf) findall 은 발견이 되더라도 계속 찾음

3. 정규 표현식
- re 모듈 (grouping)

- match나 search의 결과로 돌려준 객체의 method
- 검색 결과의 특정 부분만 출력한다
- 매치된 문자열을 돌려준다

method	목적
group()	매치된 문자열을 돌려준다.
start()	매치된 문자열의 시작 위치를 돌려준다.
end()	매치된 문자열의 끝 위치를 돌려준다.
span()	매치된 문자열의 (시작, 끝)에 해당하는 튜플을 돌려준다.

```
re.match(r'(₩d{6})-(₩d{7})', '123456-7890123')
```

```
<re.Match object; span=(0, 14), match='123456-7890123'>
```

```
m = re.match(r'(₩d{6})-(₩d{7})', '123456-7890123')
m.groups()
```

```
('123456', '7890123')
```

```
m.group(1)
```

```
'123456'
```

```
m.start()
```

```
0
```

```
m.group(2)
```

```
'7890123'
```

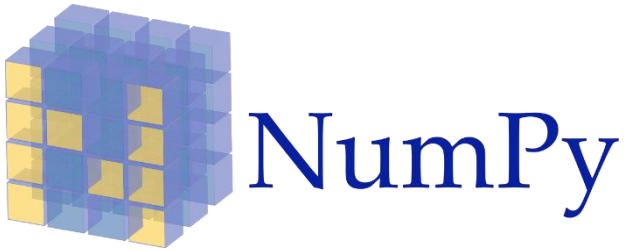
```
m.end()
```

```
14
```

```
m.span()
```

```
(0, 14)
```

4. numpy
- numpy에 대해서




```
import numpy as np
```

numpy (numeric python) 란?

: 벡터나 행렬 등 다차원 배열을 쉽게 다룰 수 있게 해주는 대표적인 라이브러리

numpy (numeric python)를 사용하는 이유

- python에서 수학/과학 연산을 위한 다차원 배열 객체를 지원한다
- 다차원의 행렬 자료구조인 ndarray 는 다양한 python 패키지의 기본 자료형으로 사용됨
- 반복적인 연산 작업을 배열 단위로 처리해 속도가 빠르며 효율적인 코딩이 가능

4. numpy
- ndarray의 생성

np.array

```
np.array([1,2,3])
```

```
array([1, 2, 3])
```

```
x = np.array([1,2,3])
print(x)
```

```
[1 2 3]
```

```
np.array([[1,2,3],[4,5,6]])
```

```
array([[1, 2, 3],
       [4, 5, 6]])
```

```
x = np.array([[1,2,3],[4,5,6]])
print(x)
```

```
[[1 2 3]
 [4 5 6]]
```

np.arange

```
np.arange(10)
```

```
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
```

```
np.arange(4,10,2)
```

```
array([4, 6, 8])
```

np.linspace

start, end, num

```
np.linspace(1,10,4)
```

```
array([ 1.,  4.,  7., 10.])
```

np.ones / np.zeros

```
np.ones((3,3))
```

```
array([[1., 1., 1.],
       [1., 1., 1.],
       [1., 1., 1.]])
```

```
np.zeros((2,4,3))
```

page, row, column

```
array([[[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]],

       [[0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.],
        [0., 0., 0.]]])
```

4. numpy
- 요소가 랜덤인 ndarray의 생성

np.random.rand

: 0과 1사이의 균일 분포 난수 생성

```
np.random.rand(5)
```

```
array([0.76171528, 0.82729454, 0.19856604, 0.2507327
4, 0.41177687])
```

```
np.random.rand(2,3)
```

```
array([[0.24736374, 0.17845016, 0.08719385],
       [0.75290982, 0.90026781, 0.56605841]])
```

np.random.randn

: 가우시안 표준 정규 분포 난수 생성

```
np.random.randn(5)
```

```
array([ 0.54383879,  0.0982076 ,  1.00668481,  0.9221
6425, -0.13896008])
```

```
np.random.randn(2,3)
```

```
array([[ 1.25912643, -0.73515639,  1.87613946],
       [-0.37457691,  0.79985258,  0.36610159]])
```

np.random.randint

: 균일 분포의 정수인 난수 생성

- high를 설정하지 않음 (범위 : 0부터 low까지)

```
np.random.randint(10, size=5)
```

```
array([9, 3, 0, 3, 2])
```

- high를 설정함 (범위 : low부터 high까지)

```
np.random.randint(10, 20, size=(2,3))
```

```
array([[17, 10, 19],
       [17, 14, 14]])
```

* 난수를 고정하고 싶다면?
- seed 값 설정하기

0

3

2


```
np.random.rand(5)
```

```
array([0.5488135 , 0.71518937, 0.60276338, 0.54488318, 0.4236548 ])
```

```
np.random.rand(5)
```

```
array([0.5507979 , 0.70814782, 0.29090474, 0.51082761, 0.89294695])
```

```
np.random.rand(5)
```

```
array([0.4359949 , 0.02592623, 0.54966248, 0.43532239, 0.4203678 ])
```

4. Numpy
- indexing과 slicing

```
arr = np.random.randint(10, 20, size=(2,3,3))
print(arr)
```

```
[[[12 11 13]
  [15 18 11]
  [18 17 18]]

 [[11 10 15]
  [14 11 15]
  [14 17 16]]]
```

- Indexing : 가리킴

```
arr[1]
```

```
array([[14, 14, 15],
       [17, 13, 16],
       [14, 13, 17]])
```

```
arr[1,1,2]
```

```
16
```

```
arr[1,1,2] = 19
arr
```

```
array([[[18, 18, 16],
        [12, 18, 17],
        [12, 11, 15]],

       [[14, 14, 15],
        [17, 13, 19],
        [14, 13, 17]]])
```

- slicing : 잘라냄

```
arr[:1,1:,:3]
```

```
array([[[12, 18, 17],
        [12, 11, 15]]])
```

4. numpy
- Boolean Indexing

Boolean indexing

- 조건문을 가지고 True, False로 Indexing 하는 것

```
np.random.seed(2)
x = np.random.randint(1,10,8)
print(x)
```

```
[9 9 7 3 9 8 3 2]
```

```
x[x > 3]
```

```
array([9, 9, 7, 9, 8])
```

```
x[x > 3] = 100
print(x)
```

```
[100 100 100   3 100 100   3   2]
```

np.where

- 조건에 맞는 값을 indexing 하여 처리
- (조건식, 조건에 맞을 때의 값, 조건과 다를 때의 값)

```
np.where(x>3)
```

```
(array([0, 1, 2, 4, 5], dtype=int64),)
```

```
x[np.where(x>3)]
```

```
array([9, 9, 7, 9, 8])
```

```
np.where(x>3, 100, x)
```

```
array([100, 100, 100,   3, 100, 100,   3,   2])
```

4. numpy
- ndarray의 재배열

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14])
```

```
b = np.arange(15).reshape(3,5)
b
```

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

a.reshape(3,5)

"Reshaping"

"Flattening"

np.ravel(b)

b.flatten()

```
array([[ 0,  1,  2,  3,  4],
       [ 5,  6,  7,  8,  9],
       [10, 11, 12, 13, 14]])
```

원본 그대로 사용

```
np.ravel(b)
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
11, 12, 13, 14])
```

copy 본 사용

```
b.flatten()
```

```
array([ 0,  1,  2,  3,  4,  5,  6,  7,  8,  9, 10,
11, 12, 13, 14])
```

4. numpy
- ndarray의 사칙연산

- shape이 같은 경우 : 같은 위치의 요소들이 계산됨

```
x = np.array([[4,4,4],[8,8,8]])
y = np.array([[1,1,1],[2,2,2]])

print(x+y)
print(x-y)
print(x*y)
print(x/y)
```

```
[[ 5  5  5]
 [10 10 10]]
[[3 3 3]
 [6 6 6]]
[[ 4  4  4]
 [16 16 16]]
[[4. 4. 4.]
 [4. 4. 4.]]
```

- shape이 다른 경우 : ValueError 발생 !!!

```
x = np.arange(10).reshape(2,5)
y = np.arange(12).reshape(3,4)

print(x+y)
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-351-ee9a50862746> in <module>
      2 y = np.arange(12).reshape(3,4)
      3
----> 4 print(x+y)

ValueError: operands could not be broadcast together with shapes (2,5) (3,4)
```

4. numpy
- Broadcasting

Broadcasting 이란?

broadcast

verb

UK /ˈbrɔːd.kɑːst/ US /ˈbrɑːd.kæst/

broadcast or US also **broadcasted** | **broadcast** or US also **broadcasted**

B2 [I or T]

to send out a programme on television or radio:

- *Radio Caroline used to broadcast **from** a boat in the North Sea.*
- *The tennis championship is broadcast live **to** several different countries.*
- *The ceremony was broadcast on the internet.*

[T]

to spread information to a lot of people:

- *I'm leaving but please don't broadcast the fact.*
- *I don't want this broadcast **to** the entire school.*

퍼뜨리다, 확장하다 !!!!

어떤 조건을 만족했을 때

- 모양이 다른 배열 간의 연산을 가능하게 함
- 모양이 부족한 부분이 확장하여 연산을 수행함
- 저차원의 배열을 고차원의 배열로 확장

4. numpy
- Broadcasting

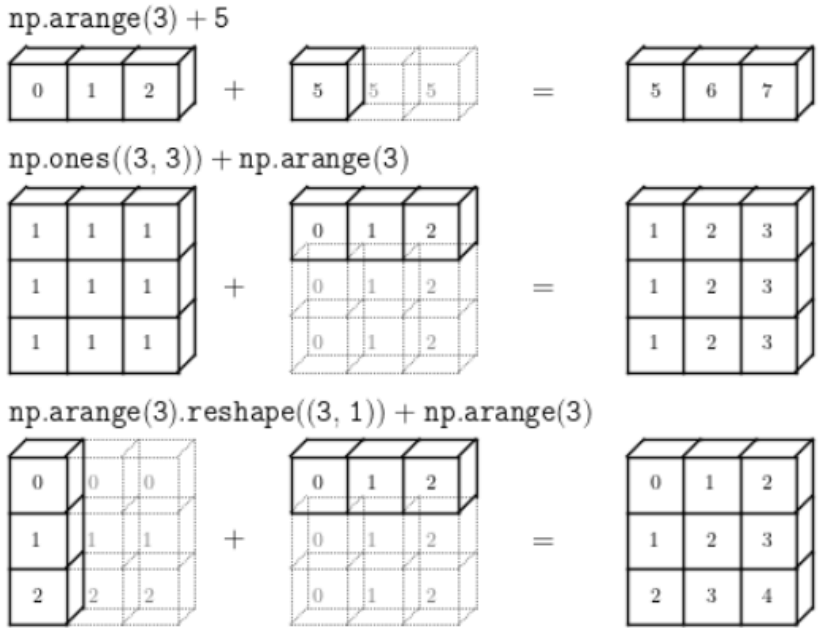
Broadcasting 조건

1) 요소가 하나인 배열은 어떤 배열에나 가능

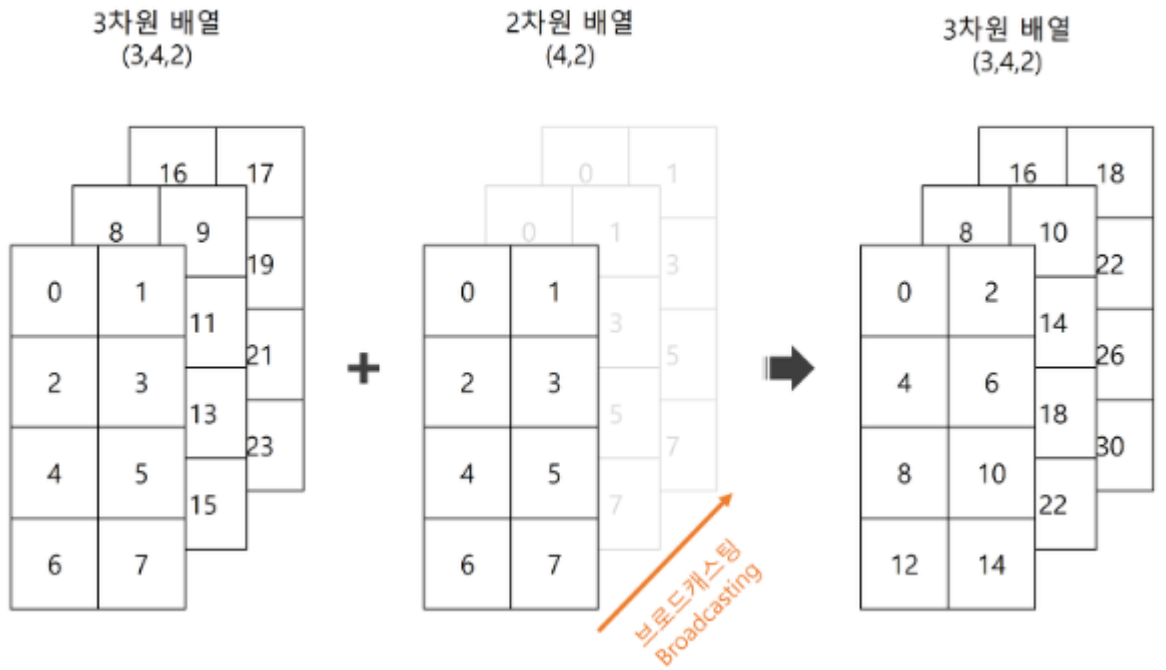
2) 뒷 차원부터 비교하여 shape이 같을 때

2) 하나의 배열의 차원이 1인 경우

3) 차원의 짝이 맞을 때



출처: *http://www.astroml.org/book_figures/appendix/fig_broadcast_visual.html*



출처 : https://sacko.tistory.com/16

질문있으신가욥 ~~~ ?

수고하셨습니다 ~~~~ !

1주차
SESSION
- 끝 -

Gram