

morning easy

---

---

---

---



## Week - 1 (문자열 처리)

"Hello, world!" 이거처럼  
문자열은  
자연 대응을 둘 다 가능

EOL: end of Line

```
"\n\n' \n" \n
' \n" \n' \n
' \n' \n' \n
but
' \n' \n' \n
```

포함하여 브레이크에 있으며 확장

+ 가능.

$$\text{char} = \backslash t \sim 821 \text{ 칸거리 한칸기 띄는 예술}$$

Enter &gt; \n = 줄바꿈.

Print(" \n\n")  
 \n  
 \n") 스스로 다음 줄로 인식  
\n\n으로  
인식되는  
줄바꿈

4. ("Red", "Green", "Blue")  $\rightarrow$  Red Green Blue  
 Print("Red", "Green", "Blue", sep=" ") 는 빈칸으로 인식  
 $\rightarrow$  Red-Green-Blue

end 풀션 기본값이 New Line

Print(" \n", end=":-\n-----\n")  
 $\rightarrow$  \n:-  
 >>

## (Week - 2 (나머지))

%A (integer) : 55, +3, 0, -4 ..

실수 - ① 2진 24진 냥수 : 1.14(592), 0.000 .. 25

$$\text{② } \frac{2.5 \times 10^{16}}{\text{floating point}} : 2.5 E(2.57 \times 10) / 10 = 2.5 \times 10^5$$

$$2.5 \times 10^5 = 2.5 \times 10^{-10}$$

실수 연산자

$$+ - * / \% \gg$$

$$\frac{1}{2} \frac{1}{3} \frac{1}{4} \frac{1}{5} \frac{1}{6} \frac{1}{7} \frac{1}{8} \frac{1}{9} \frac{1}{10}$$
연산자 우선순위  
연산자

$$9/4-3$$

$$\frac{9}{4}-3$$

$$\frac{9}{4}-3</$$

## Week - 3

• 모듈 같은거에 딱 와 같은  
여러 품목, 상품 존재.

법률와 지침

부록 : 2018년 주요 이슈

↳ <변수> = <식>

사용가능하기도 않.

지정문: 벌칙 범정

$$\Rightarrow x = ? \quad \underline{\text{[ANS]}}$$

$$\Rightarrow x = x + 2$$

다음의 2

\* • H 부분      X = input( ) 할 때

## 키보드 입체

- $\begin{cases} x=3 \\ y=7 \end{cases}$   $\rightarrow$  0점과 범수  $x$ 는  $x_0$ ,  $y$ 는  $y_0$  ...

$$\textcircled{1} \quad \begin{array}{l} x=3 \\ y=7 \end{array}$$

$$\textcircled{1} \quad x, y = 3,$$

$x, y = 4x$  : 동시 대체방법과 일시 번호 표기

● **번수 이용하기** : ① **숫자를 선택·불기**

## ② 갑의 특성을 잘 예상

## ④ 자신의 규칙을 만들고 일관성 찾기

( 관습에 따랐으 ( 일반 베수드 소문자로 시작 )

⑤ 예약금 (이미 입금한 경우)는 없다.

ex) import

주석: ~~~~~ ~~#~~ 1123이 ~

2Week - 1 핵심

round(2135, -1) = 2140

$\rightarrow$  (2135, -1)  $\rightarrow$  2140

24:12M, 00:00 ~  
9시 30분 후

• 함수 -

return

내용 (2022)

round(n, 2)  $\rightarrow$  2140

return round(area, n)

print(areaCircle(3.1))  $\rightarrow$  0.8 π  
" " (3.2)  $\rightarrow$  0.8 π × 3.141592653589793

람다식:

(lambda radius: math.pi \* radius \*\* 2)(3)  
키워드  
파라미터  
(값이 반환되는 변수)  
문자  
(값)  
연산  
↓

areaCircle = lambda radius: math.pi \* radius \*\* 2

areaCircle(8)  
areaCircle(5)  
인수 넣은 거랑  
같은 결과.  
!

• 함수 복잡화: 프로그램을 간단하게 만들기

• 함수 정의

파라미터

코드 복잡 (4줄 들어감)

def <함수 이름>(<변수>, <변수>, ..., <변수>): <문자>

람다: 익명함수. 필요의 이름을

붙여 정의하거나 변수에 대입하기 위해

함수를 훈련/보통할 때 사용하는식으로

작성된 함수.

\* lambda, def,

• 함수 호출 인수... 실제 파라미터

<함수 이름>(<식>, <식>, ..., <식>)

{  
이미 정해진  
함수만

지정

def <함수 이름>(<변수>, <변수>, ..., <변수>): <문자>

2 week - 2 논리식 ( 해석 결과가 True or False로만 나타나는 것)

True, False 초기 단위로 대응

### • 논리 연산자

논리곱: and : True and True = True, 나머지都不是 False

↑ 이용연산자, 정의하기

논리합: or : False or False = False, 나머지都不是 True

논리 역: not : True  $\rightarrow$  False, False  $\rightarrow$  True

↓ 단항연산자, 정의하기

\* 우선순위 : not > and > or

주의점

• 단축제산 : and, or은 True, False 만 빠져

결과를 알 수 있는 경우 일 때, 그때만 실행

계산을 생략하는 것

def loop():  
  $\rightarrow$  주로 반복문, 파일을 반복해서 처리할 때  
 loop()  
 그 계산은 넘으면 오류

False and loop()  $\Rightarrow$  단축제산 0  
False

True and loop()

??

### • 비교 연산자 (모든 이용연산자, 정의하기)

같다 다르다 크다 작다 크거나 같다 작거나 같다

== != > < >= <=

True == True = 1, False == 0 비트연산

단항연산자 아스키 값이 크기 기준으로 비교 (소용자가 대응하는 아스키 값을)

## 2Week - 3 조건문

```
import math

def areaCircle(radius, n):
    if radius >= 0:
        area = math.pi * radius ** 2
        return round(area, n)
    else:
        return 0.0
```

### 문법

```
if <조건>:  
    <실행>  
else:  
    <실행>
```

# 디버깅 예제: 아래 생략 가능 (return 값 없음, void)

### ④ 조건을 취는

```
if 조건:  
    실행  
elif 조건:  
    실행  
else:  
    실행  
    예외 처리 생략 가능
```

▶ 예제는 꼭 보자! 상주보다 예외!

### • 정수 조건

X = int(input("Enter the score : "))

if X % 2 == 0:

if X % 3 == 0:

print ~

else:

print ~

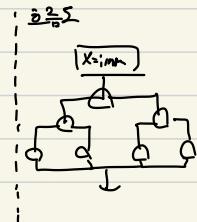
else:

if X % 3 == 0:

print ~

else:

print ~

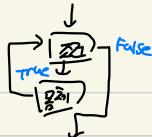


## 2 Week - 4 반복문 (loop)

ex) Class, object, ..

문법

While ~~문법~~:  
<문제>



\* while

<문제>. isdigit()  $\rightarrow$  865이 숫자로만 되어있음  
"865". isdigit()  $\rightarrow$  True  
"3650". isdigit()  $\rightarrow$  False  
"17". isdigit()  $\rightarrow$  False  
"three". isdigit()  $\rightarrow$  False

△ 예제: 입력 확인 문제

X = input()

while not <문제>:

X = input()

※

while True

X = input("Enter your number: ")

while not X.isdigit():

X = input("Enter your number: ")

X = int(X)

if X % 2 == 0:

if X % 3 == 0:

print(X, " ~ ")

else:

print(X, " ~ ")

else:

if X % 3 == 0:

print(X, " ~ ")

else:

print(X, " ~ ")

more = input("더 입력할 Y, 아니면 아까나 ")

if more != 'Y':

break

Print("Bye!")

break는 강제로 있는

기능 가능으로 빠르게 탈출

## 2 Week - 5

## index (우리 번역)

**<문자열> [<정수식>]**

ICT 풍선탐  
0 1 2 3 4  
-5 -4 -3 -2 -1

>> "ICT 융합" [2] ~ [1+1] 도 가능  
C  
중수관!

<문자열> . partition(<문자열>)

"3141592". Partition C" 0")

개념 + , 총 3 칸으로

S = "ICT 융합"

$\delta$  [q : q] ↗ 6이란 뜻은  
도난당한 물건까지 출동

Void 키워드  
Void 키워드는

Void on oltien

$\Rightarrow \text{ICT 응집}$

(("3", ":" , "14592") ) ( , - )  
1 2 3       $\frac{E_1 + E_2}{2} = 0.4745$   
Emin = 0.4602

## 튜플이라 쓸

21이어 모아놓으

류중도 기사 번역일정

(~인간) → 공식으로 나온

④ 반응원 유형은 어떤가?

### 3 Week - 1

#### 자연수

정수의 집합  $\mathbb{N}$  : 자연수 집합

$$N = \{0, 1, 2, 3, \dots\}$$

#### • 귀납법(인도수)

1) 기초:  $0 \in \text{자연수}$

2) 인도선:  $n \in \text{자연수} \Rightarrow n+1 \in \text{자연수}$

3) 그 외에 다른 자연수는 없다.

] 자연수에 대해 증명하는 방식 - 재귀함수

# Week 3 - 2 초입이 출석

요구사항: 정수 n을 인수로 받아서 n부터 1씩 줄여가면서 홀수인 10기 하나씩 Print,

def countdown(n):  
 if n > 0:  
 print(n)

홀수 인수는 오직 odd 수만. 빌새! 안 Print.

[Countdown(n)]

반복문  
 $n > 0$

1. 인수 n을 Print
2. 1sec 쉬기
3. Countdown(n-1) 반복

종단 조건  
 $n \leq 0$

빌새! Print

ex) import time

def countdown(n):

if n > 0:

print(n)

time.sleep(1)

countdown(n-1)

else:

print("빌새!")

≠ Count down 화살표를 재귀함수 X로 하는 방식 (Loop 활용)

ex) import time

def countdown(n):

while n > 0:

print(n)

time.sleep(1)

n = n - 1

print("빌새!")

자기 VS 반복  
재귀함수  
Top-down  
Bottom-up



반복 - Bottom-up

```
def Sigma(n):  
    sum = 0  
    while n>0:  
        n, sum = n-1, n+sum  
    return sum
```

ex)

```
def Sigma0(n):  
    sum = 0  
    while n>0:  
        n = n-1  
        sum = n+sum  
    return sum
```

def Sigma1(n):

```
sum = 0  
while n>0:  
    sum = n+sum  
    n = n-1  
return sum
```

$$\text{Sigma } 0(10) = 45$$

$$\text{Sigma } 1(10) = 55$$

$\Rightarrow n=10, \text{sum} = n+\text{sum}$  으로 대체

정리

- ① 하향식 적용 재귀함수! EASY 코딩 데려 공간효율↓
- ② 상향식 적용 재귀함수 on while · loop 데려 공간 비효율×
- ③ 일반 재귀함수 대체로 재귀기능으로 기계적의 반복 가능
- ④ 재귀기능 함수를 반복문으로 변환하는 것도 기계적이다.
- ⑤ 하향식 작성 → 고효율로 넘나들기

● 파이썬은 재귀제한, 디자인 흐름 제한 있음

why: 파이썬은 while, for loop를 좋아해  
010~

# Week 3-4

A21: 거듭제곱

$$n \times n = n^2$$

$$\text{pow}(n, 2) = n^2$$

$b^n$  계산해 주는 형식  $\text{pow}(b, n)$ 을 네트워크 만들기.

$b$ 는 정수,  $n$ 은 자연수 제한

정수 범위는 모두 0으로 초기화

```
def power(b, n):
    if n > 0:
        return b * power(b, n-1)  ↳ 22번 ↳
    else:
        return 1
```

```
def power(b, n):
    def loop(b, n, prod):
        if n > 0:  ↳ 22번 ↳
            return loop(b, n-1, b * prod)
        else:
            return prod
    return loop(b, n, 1)  ↳ 22번 ↳
```

```
def power(b, n):
    def loop(n, prod):
        if n > 0:
            return loop(n-1, b * prod)
        else:
            return prod
    return loop(b, n, 1)
```

↓  
while 반복

```
def power(b, n)
    prod = 1
    while n > 0:
        prod = b * prod
        n = n - 1
    return prod
```

# Week 3

기능제한 | 개수 : 흔/적은 구별개수

def fastpower(b, n):

if  $n > 0$ :

if  $n \% 2 == 0$ :

return fastpower( $b^{**} 2$ ,  $n/2$ ) ✓ 22번의 틀렸을 때

$\log_{\frac{1}{2}} n$ . 좀 더 빠름.  $n=0$ 이 되는지 체크

else:

return  $b * \text{fastpower}(b, n-1)$  ✓ 22번의 틀렸을 때 X

else:

return 1

계산 비용 분석

시간  $\log_2 n$

→ 절약됨.

공간  $\log_2 n$

22번의 틀렸을 때

def fast power(b, n):

def loop(b, n, prod):

if  $n > 0$ :

if  $n \% 2 == 0$ :

return loop( $b^{**} 2$ ,  $n/2$ , prod)

else:

return loop( $b$ ,  $n-1$ ,  $b * prod$ )

else:

return prod

return loop( $b$ , 1)

시간:  $\log_2 n$

공간: 1

반복

def fast power(b, n):

prod = 1

while  $n > 0$ :

if  $n \% 2 == 0$ :

$b = b^{**} 2$

$n = n//2$

else:

prod =  $b * prod$

$n = n-1$

return prod

## Week 4 - 0

제7주차 반복 : 정렬

data A 시를 봐가는 기준 → 정렬 (sorting)

list 구조

ex) numbers = [3, 5, 4, 2] 정렬

numbers . sort() → [2, 3, 4, 5]

numbers . sort (reverse = True) → [5, 4, 3, 2]

dict = ["길동이", "민수", "다현", "소연", "인경"]

dict . sort() → 가나다순 정렬

\* list & for loop

odd  $\rightarrow$    
 odd[2] = odd[1] + 1  
 why? : 초기 X, 2번 째 같은 위치에 저장하는 거임?  
 그 뒤에는 odd[2] = odd[1] + 1 해야 함.

Week 4. | : 시퀀스: 여러개의 data 일정한 구조로 data 그룹

Index		
리스트	[4, 6, 9, 11]	수집 가능 $\rightarrow$ odd[1:3] = [34, 44, 55] $\rightarrow$ [1, 34, 44, 55, 7, 9] 7번 째는 <del>odd[0]</del> ok
튜플	('김동현', 1, '중')	$\rightarrow$ t[0]+t[1] = '김동현' + '중' X. s = t[0]+t[1] = '김동현 중' but 4번 째 X
정수범위	range(3, 9)	수집 불가능 range(s) = [0, 1, 2, 3, 4], n = range(s) $\rightarrow$ 8 in n: True, 5 in n: False range(3, 9) = [3, ..., 9] range(3, 11) = [3, 5, 7, 9] $\rightarrow$ 정수 시퀀스 가능 range(11, 2, -2) start > end
텍스트 시퀀스 타입	문자열 String	"김동현과학"

시퀀스 연산 (S: 시퀀스 / x: 원소 / i, j, k: 인덱스 / n: 크기)

x in S : x가 S에 있는지 True, 없으면 False

x not in S:  $\neg$

S[i]: S의 i 위치에 있는 원소

S[i:j]: 위치 i ~ j-1 까지의 원소

S[i:j:k]: 위치 i ~ j 까지 k 간격으로 뽑은 시퀀스 조작 (i는 제외)

len(S): S의 크기. 리스트 S의 원소 개수

min(S): S의 가장 작은 원소

max(S): 가장 큰 원소

S.index(x): S의 가장 앞에 나오는 원소 x의 인덱스 위치번호

S.index(x, i): S의 위치 i에서 시작하여 가장 앞에 나오는 원소 x의 인덱스 위치번호

S.index(x, i, j): S의 위치 i로부터 위치 j 깊까지에서 가장 앞에 나오는 원소 x의 인덱스 위치번호

S.count(x): S에서 원소 x의 빈도수

S+t: S와 t 나란히 붙이기

S\*n: S를 n번 반복하여 나란히 붙이기

n\*S

!!

## Week 4-2

for 루프:

이상의식이 맞는동안 → ~~종료조건~~, return, break를 만나면  
문자를 반복하거나  
문출가능

for <변수> in <시퀀스>:  
<문자>  
:

30일로 구성된 달

for month in [4, 6, 9, 11]:

4  
6  
9  
11

for i in range(5)  
Print(i)

0  
1  
2  
3  
4

for i in range(10, 3, -3)  
for j in range(3, 11, 3)  
Print(i, j)

10 3  
10 3  
10 9  
7 2  
7 2  
7 9  
4 3  
4 6  
4 9

```

import time
def Countdown(n):
    while n > 0:
        print(n)
        time.sleep(1)
        n -= 1
    print("Blastoff!")

```

```

import time
def Countdown(n):
    for i in range(n, 0, -1):
        print(i)
        time.sleep(1)
    print("Blastoff!")

```

{ def Sigma(n):

Sum = 0  
while n > 0:

n, Sum = n-1, n+sum

return Sum

def Sigma(n):

Sum = 0  
for i in range(n, 0, -1):  
 Sum += i

return Sum

\* 두가지 2분정도 문제에서 나중에 드가

## Week 4.3 listy 7/14/22

리스트	VS	배열
리스트		불가능
배열		가능
		고정

## 파악은 어느?

○ 어느 정도 가능 (F번)

~) 길이를 2배로 늘어나는 배수이라면

## 리스트 구조구성

1) **가능** : 배열 [ ] 은 리스트이다.

2) 기능:  $X$ 가 임의의 원소이고  $XS$ 가 임의의 리스트라면,  
 $X :: XS$  도 리스트이다.

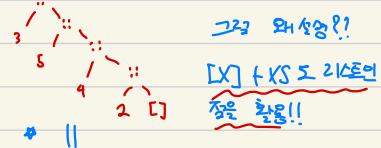
여기서, :: 는 리스트 생략 연산자로 Constructor를 제거한다.

$X^L$ 은 선두원소(head),  $X^S$ 은 후미리스트(tail)라고 한다.

그 외에 다른 리스트는 있다.

ex) [3,5,4,2] : 3 :: (5 :: (4 :: (2 :: [])))

파이썬은 자연X



$$\cdots [5] + ([4] + ([2] + [1]))$$

이 고장으로 만들었다고 가장

# Week 4-4

## 선택정렬

정수 list ns를 선택정렬 하려면

반복조건 ✓	ns != []	<ul style="list-style-type: none"> <li>ns에서 가장 작은 수를 찾아서 smallest에 넣기,</li> <li>이후 ns에서 제거</li> <li>ns를 차례로 선택정렬하기</li> <li>smallest와 ns를 차례로 나누어 넣어주기</li> </ul>	
		증강조건 ✓	<p>증강상황 원소가 있으므로 []를 그대로 반환한다.</p> <p>전체 정렬 코드 예시</p>

### • list 처리 메소드

ns.remove(n) : ns에서 가장 앞에 나오는 원소 n을 제거한다.

list ns에 접근 ValueError 발생

### • append

ns.append(n) : ns의 맨 뒤에 n을 넣는다.

def selectionSort(chs):

if ns != []:

smallest = min(ns)

ns.remove(smallest)

return [smallest] + selectionSort(chs)

else:

return []

↓ 풀이자리

def SelectionSort(chs):

def loop(ns, ss):

if ns != []:

smallest = min(ns)

ns.remove(smallest)

return loop(ns, ss + [smallest])

else:

return ss

return loop(ns, [])

↓ while

def SelectionSort(ns):

ss = []

while ns != []:

smallest = min(ns)

ns.remove(smallest)

ss.append(smallest)

return ss

# Week 4-5

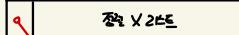
## 삽입 정렬

정수 list ns 를 삽입 정렬 하려면

반복조건

ns = []

- ns의 후미리스트인 ns[1:] 를 차귀로 삽입 정렬 하여 ss2 를 찾.
- ns의 선두원소인 ns[0] 를 정렬된 list 와의 학합은 q22 를 찾고, 리턴한다.

ns 

제 위치의  
기여 등을

q22 O 2스트

↓ ex

정렬조건

ns == []

정렬할 원소가 없으면 [] 를 그대로 리턴한다.

• 4-5. 4월 29일 insert 설명

자기本身 삽입 정략

insert 세계 정략

① 정렬된 리스트 ss의 : insert(1, [2,4,5])  
선두원소보다 큰 수 가능해지기 > [1] + [2,4,5]

② 정렬된 리스트 ss의 : insert(3, [2,4,5])

선두원소보다 큰 수 가능해지기 > [2,4,5] + insert(0, [4,5]) 차이

③ 빈 리스트에 가능해지기 : insert(1, [])

[1]

↓  
세부 설명

insert!!

정수 n 을 정렬된 list ss의 제 위치에 기여 넣으려면

반복조건

ss = []

- ss 를 선두원소 ss[0] 와 후미리스트 ss[1:] 로 나눈다.
- $n \leq ss[0]$  이면, n 을 ss 의 앞에 넣는다.
- $n > ss[0]$  이면, 차귀로 n 을 ss[1:] 의 제 위치에 기여해준다.  
그 앞에 ss[0] 을 넣는다.

def insert(n, ss):  
if ss != []:

if n <= ss[0]:  
return

else:  
return

else:  
return

22(자기本身)

def insert(n, ss):  
def loop(ss, left):

if ss != []:

if n <= ss[0]:  
return  
else:  
return

else:  
return

return loop(ss, left)

TIP1: loop의 두 째 파라미터 left 는

n 은 정렬하기 불필요한 list 를 나타낸다.

아래 실습 속도 차이를 줄이기 때문이다.

TIP2: n 은 기여 값을 차례로 활용으로 left, n,

ss 속으로 나열되며 list 는 불필요, 비용.

while

insert sort

|| 22(자기本身)

|| while

|| for

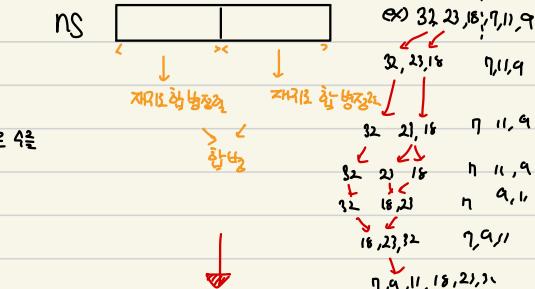


## 합병정렬

784 list ns를 합병정렬 실행

		NS	<table border="1"><tr><td></td><td></td><td></td></tr></table>				32, 23, 15, 7, 11, 9
반복 조건	len(ns) > 1	• NS를 빈으로 초기화, 각각 자리를 합병정렬 원본.		32, 23, 15, 7, 11, 9			
	len(ns) <= 1	• 두 정렬된 리스트를 알맞은 위치로 차례대로 흘려가며 가장 작은 수를 먼저 선택하는 방식으로 하나로 합병하여 만들고		32, 23, 15, 7, 11, 9			

정렬된 풀이가 있으므로 그대로 기반



```
def mergesort(ns):
    if len(ns) > 1:
        mid = len(ns)//2
        return merge(mergeSort(ns[:mid]),
                    mergeSort(ns[mid:]))

    return ns
```

```
def merge(left, right):
    if not left == [] or right == []:
        if left[0] <= right[0]:
            return [left[0]] + merge(left[1:], right)
        else:
            return [right[0]] + merge(left, right[1:])

    else:
        return left + right
```

append  
[ ](!) 만족기록

??(자기)

Swhite

4-7

크루저

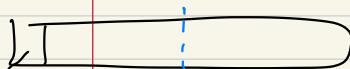
포인트: 7점

↳ 피봇 값을 기준으로 짐을 가 모조의 원형, 큰 다 모양

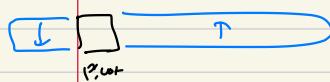
증상 /list n 크루저

반복 조건	$ cn(cns)  > 1$	<ul style="list-style-type: none"> <li>기준으로 사용한 피봇값 하나다. 편의상 맨앞에 있는수 선택</li> <li>Pivot 기준 - 짐 - Pivot - 짐</li> <li>원, 오른쪽 각각 차례로 짐을 끌어올 때 2-1, 가운데로 Pivot 기준으로</li> </ul> <p>2번</p>
증상 예제	$ cn(cns)  = 1$	그대로 2번

o 리스트가 반복 조건하는 Pivot 값의 제일 중앙.



Pivot



Pivot

5-0

## 자구(자) 반복: 검색

- 시퀀스 검색

연산	의미
<code>s.index(x)</code>	시퀀스 <code>s</code> 에서 가장 앞에 있는 키 <code>x</code> 의 위치번호를 리턴
<code>s.index(x, i)</code>	시퀀스 <code>s</code> 의 <code>i</code> 위치에서 시작하여 가장 앞에 있는 키 <code>x</code> 의 위치번호를 리턴
<code>s.index(x, i, j)</code>	시퀀스 <code>s</code> 의 <code>i</code> 위치와 <code>j</code> 위치 사이에 가장 앞에 있는 키 <code>x</code> 의 위치번호를 리턴 ( <code>i</code> 위치는 검색 범위 O but <code>j</code> 는 검색 범위 X)

ex) `s = [36, 74, 25, 98, 13, 86]`

`s.index(74)`  $\Rightarrow$  1

`s.index(86)`  $\Rightarrow$  0 (두번쩨 일기도 같은 원인)

`s.index(36, 3)`  $\Rightarrow$  5 (36에서 시작)

`s.index(36, 3, 6)`  $\Rightarrow$  ValueError, 같은거 검색

if  $\overset{2\text{번}}{(x)}$  in `s`: `s.index(x)`

$\rightarrow$  있으면 리턴, 없으면 끝내기

# 리스트 검색. O/X 문제

o 입출력 요구사항

입력: 리스트 S, 키 key

출력: key가 S에 있으면 True, else: return False

↓

리스트 S에서 key 검색

반복문	S != []	So의 첫번째 원소 S[0]가 Key와 같으면, → return True 그렇지 않으면, So의 뒤에 있는 S[1:]에서 key를 재귀로 검색
증명	S == []	검색 대상이 없으므로 False를 return

```
def seqSearch OX (S, key)
```

```
if S != []:
```

```
    if S[0] == key:
```

```
        return True
```

```
    else:
```

```
        return seqSearch OX (S[1:], key)
```

```
else:
```

```
    return False
```

S while

```
def seqSearch OX (S, key):
```

```
while S != []:
```

```
    if S[0] == key:
```

```
        return True
```

```
    else:
```

```
        S = S[1:]
```

```
return False
```

S for loop

```
def seqSearch OX (S, key):
```

```
for x in S:
```

```
    if x == key:
```

```
        return True
```

```
return False
```

S-2

# 이분검색 알고리즘

.sort() 사용

검색횟수 줄임

정렬된 list SS와 key를 정의

반복문	SS = []	SS의 정 가운데 원소의 위치번호는 mid 2자
		key가 SS[mid]와 같으면, 찾았으므로 True return 작으면, SS[:mid]에서 key를 재귀호출 크면, SS[mid+1:]
종료조건	SS == []	검색 대상 없으면 False return

```
def binSearchOX (SS, key):
```

```
if SS != []:
    mid = len(SS)//2
    if key == SS[mid]:
        return True
    elif key < SS[mid]:
        return binSearchOX (SS[:mid], key)
    else:
        return binSearchOX (SS[mid+1:], key)
```

```
def binSearchOX (SS, key):
```

```
while SS != []:
    if key == SS[mid]:
        return True
    elif key < SS[mid]:
        SS = SS[:mid]
    else:
        SS = SS[mid+1:]
return False
```

5-3

## 2| 스택 검색 : 주소식 접근 |

입출력 요구사항

입력 : list S, ≠ key

출력 : Key가 S[ki] 칸을 찾았을 때 index를 반환  
없으면 return None

(순차검색방법) -----

```
def seqSearch(S, key):
    i = 0
    for X in S:
        if X == key:
            return i
        i += 1
    return None
```

```
def seqSearch(S, key)
    for i in range(len(S)):
        if S[i] == key:
            return i
    return None
```

2번 세 줄은 지우기

이분검색 알고리즘

```
def binSearch(S, key):
    low = 0
    high = len(S) - 1
    while low <= high:
        mid = low + high // 2
        if key == S[mid]:
            return mid
        elif key < S[mid]:
            high = mid - 1
        else:
            low = mid + 1
    return None
```

예산	트리미
시퀀스 Population or k	
중복 없이 K개를 무작위로 뽑는	
리스트로 return	
Random.sample(n)	Random.sample(n)
정수 하나 무작위로 뽑는다. return	정수 하나 무작위로 뽑는다. return



import random

random.sample(range(10000), 100)

S

1 ~ 10000 까지 100개 랜덤으로 고르기

↳ H2

db = random.sample(range(1000000, 10000000), 100)



def testSeqSearch():  
 print("Sequential search function test")  
 for \_ in range(10):  
 key = random.randrange(11000000)  
 print(key, "found at", seqSearch(db, key))

이번 티어는  
실습코드  
→  
key 찾기  
중간값을  
찾아내는  
방법

def testBinSearch():  
 print("Binary search function test")  
 db.sort()  
 for \_ in range(10):  
 key = random.randrange(11000000)  
 print(key, "found at", binSearch(db, key))

# 파일 입출력

5-4

파일

File

~ 정보 영구 보존 방식

Text File

~ 키보드 입력 가능한

문자열만 있는 파일

- ① 파일 만들기 파일이면 쓰기 모드
- ② open('article.txt', 'r') read 모드, 읽기 용
- ③ f.close() [연락처 다쓰면 끝나야 함. 보면 데려]

파일 접근 모드

`f = open('output.txt', 'w')` 쓰기  
`f.write('z')` ←  
`f.close()` in: 끝나고 해야 함

모드	의미
"r"	파일에서 읽음 (파일이 없으면 오류)
"w"	파일기록 (파일이 있으면 지우고 새롭음, 없으면 만듬)
"a"	파일의 뒤에서 이어서 쓸 (있으면 만듬)
"r+"	파일에서 읽고 쓸 (파일이 없으면 오류)
"w+"	파일이 쓰고 읽음 (파일이 있으면 지우고 새롭음, 없으면 만듬)
"a+"	파일의 뒤에 이어서 쓰고 읽음 (있으면 만듬)

↓ 읽기

`t = open('input.txt', 'r')` 파일 한줄씩 모드  
`print(t.read())` → 1줄자 read 파일  
`t.close()` t.read() 한번 다읽음.  
readline 하면 줄단위로 할  
한줄에서만

## 파일 메소드

메소드

의미

close()

파일 닫음

read(n)

파일의 현재 위치에서 문자 n개를 읽어서 문자열로 return

read()

"" 파일의 끝까지 모든 문자열로 return

readline(n)

"" 그 줄의 문자 n개를 읽어서 문자열로 return  
문자열 n개

readline()

"" 그 줄의 끝까지 모든 문자열로 return

readlines()

"" 한 줄씩 끝까지 읽어서 줄의 list로 return

write(s)

문자열 s를 파일의 현재 위치에 쓴다

write(lines(ss))

"" list ss가 있는 문자열을 모두 파일의 현재 위치에 쓴다

# 5.4

## 텍스트 파일에서 문자열 검색

### \* 문자열 메소드 (마지막 영상 2분부터 시작)

메소드

의미

str.find(sub) str'에서 맨 앞에 나오는 'sub'의 시작 위치번호를 내줌, 있으면 '-1'을 내줌

str.index(sub)

||

, 있으면 'ValueError' 오류

str.rfind(sub) str'에서 맨 뒤에 나오는 'sub'의 시작 위치번호를, 있으면 '-1'을 내줌

str.startswith  
(prefix) str이 Prefix로 시작하면 True를, 그렇지 않으면 False를 내줌

str.endswith  
(suffix) str이 Suffix로 끝나면 True를, 그렇지 않으면 False를 내줌

[마지막 영상 4분쯤부터]

→ 글씨 있는가 찾기

```
def findFirst(filename, key):
    infile = open(filename, 'r')
    outfile = open("result.txt", "w")
    text = infile.read()
    text.find(key)
    if position == -1: → 안기 때문.
        outfile.write(key + " is not found.\n")
    else:
        outfile.write(key + " is at " + str(position) + ".\n")
    outfile.close()
    infile.close()
    print("DONE")
```

findFirst('article.txt', 'Computer')

⇒ » DONE

[마지막 연습 4. 블록부트캠프]

-문서에 있는가 찾기

```
def findSecond (filename, key):
    infile = open(filename, 'r')
    outfile = open ("result.txt", "w")
    text = infile.read()

    position = text.find(key)
    position = text.find(key, position + 1)

    if position == -1:
        outfile.write(key + " is not found.\n")
    else:
        outfile.write(key + " is at " + str(position) + " the second time.\n")

    outfile.close()
    infile.close()
    print ('DONE')
```

6week - 재구와 반복 : 트리재귀

↓  
제산 대상이 족발적 증가 규모로

- 피보나치수열 (조합제산)

→ 하강식 증등. 족발

- 하노이 탑

상향식의 경우 중복 고정 ↓

# 6 week - 1 피보나치 수열

	0	1	2	3	4	5	6	7
i								
아기토끼	0	1	0	1	1	2	3	5
어른토끼	0	0	1	1	2	3	5	8
$fib(i)$	0	1	1	2		3	5	8
	1	2	3	5	8	13	21	34
	55	...	...					

$$fib(n) = fib(n-1) + fib(n-2), n \geq 1$$

$$fib(1) = 1$$

$$fib(0) = 0$$

) 재귀식

↓

def fib(n):

if  $n > 1$ :

return  $fib(n-1) + fib(n-2)$

어느정도 예상은 했지만↑

else:

return  $n$

→ 재귀호출.

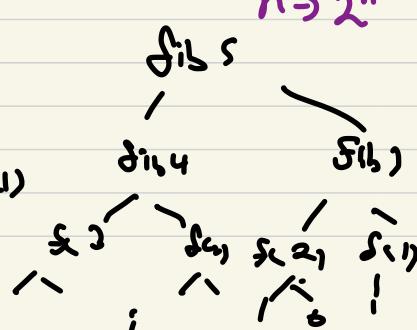
$fib(5)$

→  $fib(4) + fib(3)$

→  $fib(3) + fib(2) + fib(1)$

→

:



# 동적계획법 (상향식)

bunny<sub>i</sub> = rabb<sub>i-1</sub>

rabb<sub>i</sub> = rabb<sub>i-1</sub> + bunny<sub>i-1</sub>

fib<sub>i</sub> = bunny<sub>i</sub> + rabb<sub>i</sub>

- - - - -

def fib(n):

~~x i=1~~

bunny, rabb = 1, 0

while i < n:

~~x i+=1~~

bunny, rabb = rabb, rabb + bunny

→ 매우 fast.

for i in range(1, n)

(n번 반복)

3번 i가 필요없음

X 81번 i는?

return bunny + rabb

for n = n

중복 계산X. 속도를 향상

S

fib(i) ←는 리스트 출력.

초반 2개 계산값 + 그다음값 = 다음값

def fibseq(n):

fibs [0, 1]

for k in range(2, n+1) #2, 3, ..., n

fibs.append(fibs[k-1] + fibs[k-2])

return fibs

6 ~~ex~~ 7월 2주  
ex-2

#n, r은 자연수인 경우

조합  $nCr$

0	0, 2, 3, ..
1	
2	
3	
..	

def Comb(n,r):

if  $r! \approx 0$  and  $r!=n!$ :

return Comb(n-1,r-1) + Comb(n-1,r)

else:

return 1

정수↑, 정수↑

파스칼 계단을 이용해 빠르게 쟁여내는 법

$$\begin{array}{ccccccccc} & & C_1 & & & & n-r+1 \\ & 1 & 1 & & & & 1 & 1 & 1 \\ & 1 & 2 & 1 & & 2C_1, 2C_2 & 1 & 2 & 3 \\ & 1 & 3 & 3 & 1 & 3C_1, 3C_2, 3C_3 & 1 & 3 & 6 \\ 1 & 4 & 6 & 4 & 1 & 4C_1, 4C_2, 4C_3, 4C_4 & 1 & & \\ & : & & & : & & & & \dots \\ & & & & & & & & \\ & & & & & & & & \end{array} \Rightarrow \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & \dots \\ 1 & 2 & 3 & & & \\ 1 & 3 & 6 & & & \\ 1 & & & & & \dots \\ & & & & & \end{pmatrix}_{n+1}$$

def Comb\_Pascal(n,r):

matrix\_x = [[0]] \* (n-r+1)

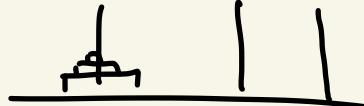
matrix [0] = [1 \* [r+1]]

for i in range(1,n-r+1):

matrix\_x[i] = [1]

```
for i in range(l, n-r+1):
    for j in range(l, r+1):
        new_value = matrix[i][j-1] +
                    matrix[i-1][j]
        matrix[i].append(new_value)
return matrix[n-r][r]
```

# 6주차 3 - 타워 오브 혼



def hanoi\_tower( $n$ , source, <sup>기준</sup> start, <sup>목적</sup> destine, middle)  
 if  $n > 1$ :

hanoi\_tower( $n-1$ , source, middle, destine)

print ("move from", source, "to", destine)

hanoi\_tower( $n-1$ , middle, destine, source)

else:

print ("move from", source, "to", destine)

원판 바꾸는 순서 등. 횟수:  $n$

$$1 \quad 1$$

$$2 \quad 1+1=3$$

$$3 \quad 3+1+3=9$$

$$4 \quad 9+1+9=27$$

$$5 \quad 27+1+27=81$$

$$n \quad (2^n - 1) + 1 + (2^{n-1} - 1)$$

$\Delta^{2n} \neq 6$  : 노드 개수 2n인 트리의 줄여나누는.

①  $\frac{1}{2}$  : odd node

②  $\frac{1}{3}$ , even node

④  $n-1$

최소반복 회수를 구하는 알고리즘  
↓  
재귀형, 외부형.

직관적인 줄이도 가능.  $\frac{1}{8}, \frac{1}{2}, n-1$  형태로 풀어쓰기

모든 나누기로 가능한 사용.

but 어떤 줄은 경우도 일치는 예외 X.

[ $10 = 5 \Rightarrow 4 \Rightarrow 2 \Rightarrow 1$  : 2가 아니라고, 풀기하기로]

$10 = 9 \Rightarrow 3 \Rightarrow 1 \rightarrow$  예외로 빼는.

② ③ ④ ⑤ ⑥ ⑦

Ex) def minStepSO(n)

if  $n > 1$ :

$r = 1 + \text{minStepSO}(n-1)$

→ 매우 오래 걸리다

if  $n \% 2 = 0$ :

$r = \min(r, 1 + \text{minStepSO}(n//2))$

if  $n \% 3 = 0$ :

$r = \min(r, 1 + \text{minStepSO}(n//3))$

return r

else:

return 0

5

중복 계산을 피하기 위한 방하고

계산값을 기억해 두고 다음에 필요할 때 가져온다.

길이가 n인 리스트의 경우

앞서 list [0]은 만드는 걸 코드를 깨끗이 하려고. 브레이트임. 즉 n개로 in  
길이 n+1이

def ministers\_l (n):

memo = [0] \* (n+1)

def loop(n)

if n>1:

if memo [n] != 0:

| return memo [n]

else:

memo [n] = 1 + loop(n-1)

if n%2 == 0:

memo [n] = min (memo[n], 1 + loop(n/2))

if n%3 == 0:

memo [n] = min (memo[n], 1 + loop(n/3))

return memo [n]

else:

return 0

return loop(n)

else [ :: -1 ] loop  
or  
212121



Reverse by 212121

## 7. Sudoku

### 1. 초기의 4 생성

```
import random
```

```
random.randint(1, 9)
```

1 to 1 (시작할 때마다 같은 숫자는 안나옴)

↓

```
ns = [1, 2, ..., 9]
```

random.shuffle(ns) ~ ns의 값이 정의로 섞임

집합 (Set) : 순서, 중복 X인 값의 집합.

```
digits = {1, 2, ..., 9}
```

```
for x in digits: print(x)
```

3 in digits

true

len(digits)

9

1  
2  
3  
4  
5  
6  
7  
8  
9

set() digits.remove(4)  
{1, 3, 5, ... 9}

positions = set()

11. add((1, 2))  
11. add((1, 4))  
↓

positions = {(1, 2), (3, 4)}

마지막에 87446 7~2의 이나온

가운데 4자, 세로 4자, 2자

S	:	1	2	3	4				
-	-	-	-	-	-	-	-	-	-
1		4	3	.	.				
2		-	1	.	4				
3		.	-	-	.				
4		.	.	4	3				

가장먼저 : 입자

세로줄 : "

수 : 2

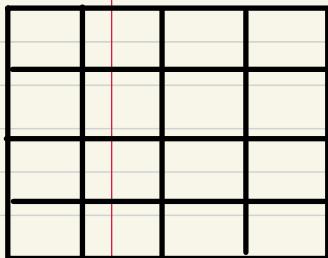
빈칸이 아닙니다. 빈칸이 아닙니다.  
자연수

빈칸이라는 틀린 소리를

빈칸은 다 차워지겠고,  
초록색을 풍깁니다  
초록색 풍깁니다

입력하는 경우 그 숫자는 빠져나와  
'22' 아닙니다. 단지에 빠져나온  
숫자입니다.

## 83. 피니 수족 알고리즘.



① 0이 0이 완성된 정답 보드 있음

$$[[3, 4], [2, 3], [1, 2], [0, 1]]$$

↓

$$X = [[0, 1], [1, 2]] \rightarrow X = [[0, 1], [1, 0]]$$

$$X[1][1] = 0 \quad \text{이거}$$

① 목적의 정답 보드 만들 = Solution

② 난이도 정하기 힘 (빈칸 개수) (no. of holes)

③ Solution과 둘을 넣어보면 틀리면 만들 (puzzles)

④ 1 no. of holes 만족 목적의로 빈칸 만들

puzzles

⑤ 빈칸의 (가로줄, 세로줄) 좌표는 집합으로 모아서 기록해둘 (holeset)

⑥ puzzles을 실행 중에 빠져나온 빈칸의 줄을 끌어

⑦ 다음 절차는 no. of holes 가 0이 될 때까지 하는 끝

A. 소자는 놓을 위치를 번호 i, 세로줄 번호 j 를 사용하여 험난함을

B. (i, j)가 holeset에 없으면 그위치 놓음

C. 빙간인 경우, 소자 (1~4)는 없애는다.

D. 이 곳에서 solution [i][j]과 같으면 puzzle이 [i][j]가

그 숫자는 놓고 장난하는 puzzle 를 풀었을 때

6:09 PM 차이연 운영.

곧 날짜가나 계속 고정일 7월에 올라온다.

7-4 214주 2주 구간. 영상 보조서 있는들기

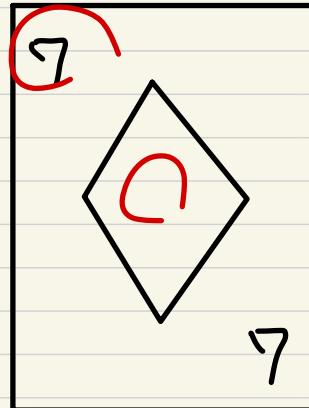
Class, Object

↳ (주체, 풀체)

ex

속성: object 고유의 특성, 상태  
(attribute)

행위: object 가 고유로 갖는 기능  
(behavior)



속성: 7, 데이터, 앞면  
특성 | 상태

행위: 두렵기

object Data Object

attributes 필드변수 (Field variable),

behavior는 메소드 (method)로 정의함.

속성 attribute	행위 behavior
특성/상태 Field variable	기능/능력 method
숫자 숫자	합수 / 조건식 함수 / 조건식

Field Variable

Suit = 'Spade'

Rank = '7'

face\_up = True

Method

def Slip():

face\_up = not face\_up

간호의 - 대각선 노선

등장 - 볼트와 금속

등장 - 제산 가능한 대체재, 수리망지 자동생성 모드

등장 - 용융재(노트)을 석화재가 제작

○ 컴퓨터 과학이 예술 / 예술

컴퓨터 : 보편 만능의 기계

| 20세기 4자리 숫자로 끝이 짜절리는 과정에서 4자리 부산물

컴퓨터의 목적 : 투명기계 , 컴퓨터의 노년상 : 투명성

불완전한 정의 : 기계적인 방식으로든 수식으로든 시스템에 걸쳐 읽을 수 있다.

투명기계 (+ 한 컴퓨터)의 특징 : 자연수의 개수를 능지 못한다.

주제 부울 : 고온, 또는, 아니 3가지로 표현.

스위치 모드 + 부울 논리  $\rightarrow$  디지털 논리

논리 모드 (부울 논리와 같은거).

디지털 : 구분해 셀 수 있는 대상

비트(bit)  $\Rightarrow$  binary digit : 2진 구조기능 (8비트 byte)

부울식



속내용 강조기 : 2 속의 0이나 만들기 전자도 모리고 사용하는게 불편 없기  
준비하는 것

차록차록

쌓기 : 속내용 강속기가 모드제작기의 준비단계.

↳ 단순연산 계산x로 1945년 개발.

## • 끝 노이만

EDVAC : 컴퓨터 초기 1945. 실제 운용 1952)

튜닝제작 능력화의 X 디자인

독립 아시아트로 1948년에 디자인 개시 대체 (마스터먼트) 이미

마이크로스토리 모드 I 이 완성됨

1945년 EDVAC 아시아트가 개시로 독립이 1950년에 ACE라는 첫 제작함.

↳ EDVAC 보다 빠르나온다. ⇒

• 왜 컴퓨터는 전기로 쓸까? → 전기 소모량을 사고 많이 주기 필요일까

만들 수 있어서

• 한사람의 뇌 안에는 2014년까지 전시기 컴퓨터의 뇌가

기전 소모를 모두 합친것보다 많은 소수가 있다.

• 글씨자는 마을의 노고, 다른 방식은 지어봐 앤

• 알고증 : SW(튜닝기)로 만드는 문제 푸는 냉도(문제풀이 냉)

↳ 냉도 : 알고증을 실행이 옮겨는데 드는 비용

▶ 예상과 : 컴퓨터로 풀기 쉬운 문제의 경우, 이 경계 넘어서 'P가 아니다' : 어떤 문제

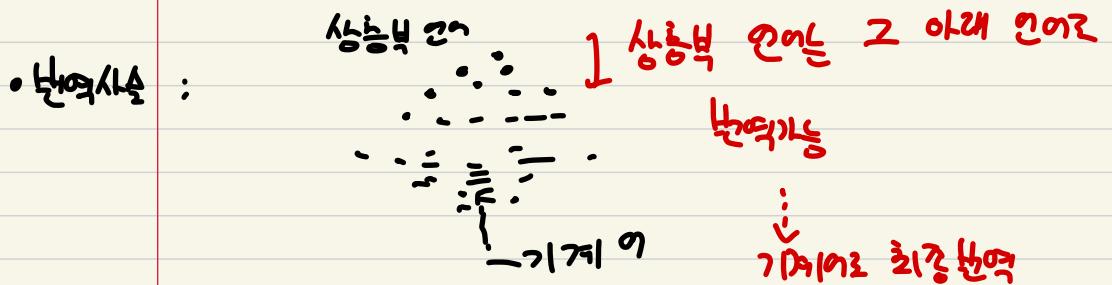
▷ 예상과 : 운 좋으면 잘하는 비용이 있는 문제를

양자컴퓨터 : 전기소자와 대신에 원자 내부의 양자를 이용해  
튜닝의 결론, 구현

↳ 튜닝의 정의는 기계적인 계산의 냉도가 NlogN진 앤드,

속도, 예측나능 그 냉도 자체로

JW 만드는 작업: 프로그래밍  
└ 사용하는 언어: 프로그래밍 언어



C9 → ML → Python, Scala → Java → C → Dalvik → 71719

튜닝 - 혼전: 투팅(제작)기능과 이언어로 디컴파일(서로 다른 언어가 번역이 되는가)

. 프로그래밍 언어 사용의 한계는 항등자 등으로 가능

└ 즉, SW가 SW를 번역가능

↓  
번역 SW를 구현해야 할 출발어와 도착어 한쌍마다 번역기를 만들 A 있음. 꿈과같기는 출발어로 코드를 제공해 번역해 도착어를 내놓음.

↳ 번역 수준이요? 컴퓨터 언어 모든 문장 부록의 의미가 하나로 끝나고 있어서. (자연어는 '인사과정의 인생도' 같이 같은 한시안에 몇몇 등장. 컴퓨터는 그니까 X)

• 컴퓨터 언어 사용의 자동 번역의 단순화 원리

① 부록 9-7 전치어 : 조립식 · 전치어 번역 결과 = 그 부분의 번역 결과

② 블록(성질 유지하기) : 부분들의 번역 결과가 항상 유지하는 성질

이해/구현방법 : 컴퓨터가  $SL\frac{1}{2}$  자동 실행하는 작업

└ 번역사슬 맨 첫바탕 언어만 대상으로 한 질문은 예제

컴퓨터: 번역사슬의 맨 일바닥 언어의 실행기

상위 언어에도 실행기가 윤리적인 소유권 등으로 구현되며 고장 가능.

◦ 첫째 하는 사람의 임대 계산하고 고안

└ 컴퓨터, 서계/약속 언어와 같은는 종종 양면일 뿐 같은 것이다.

# 정류장 인식

- 인근 지능형 학장
- 인근 네트워크 학장
- 인근 예상학장 이정표.

반올림 .

round ( $x, n$ )  
 $x$ 의 소수 부분까지  $n$ 자리로 반올림

Sum: 21이 안에 있는 더하기. sum([1, 2, 3]) = 6

max: 21이 안에 있는 최대값 max([4, 2, 1, 5]) = 5  
min: 21이 안에 있는 최소값 min([-3, 1, 0, 2]) = 0

" "

abs(-x) = x. 절대값

if  $x : \sim\sim$   
↳ 반복문 구현하기  
줄정.

$n * m = n^m$

$n * m = nxm$

$n / m: \frac{n}{m}$

$n // m: \frac{n}{m}$ 의 몫

$n \% m: \frac{n}{m}$ 의 나머지

for i in  $\sim\sim$   
↓  
0 ~ 숫자 - 1 까지

for i in list

↓

list [0] ~ list [-1] 까지

