본 컨텐츠는 과학기술정보통신부 정보통신기획평가원의
SW중심대학지원사업단의 연구결과로 수행되었습니다.
(2019-0-01838)

과학기술정보통신부 IITP 정보통신기획평가원 배재대학교 PAI CHAI UNIVERSITY AI·SW중심대학사업단

파이썬 기초와 게임 만들기 - 2

Taejun Lee (marlrero@kakao.com, +82-10-5223-2912) Doctor's course
Paichai University, Dept. Computer Engineering, Lab. MIE

2023 동산고등학교-배재대학교 동아리 교육

CONTENTS

01

정수와 실수

1. 저장 방식
2. 산술 연산자
3. float와 int
4. 복합 대입 연산자
5. 소괄호

1-1. 저장 방식

- 정수 저장 방식과 실수 저장 방식
  - 정수 2 이상 5 이하의 모든 정수를 저장하라!
    → 2, 3, 4, 5 저장 가능
  - 실수 2.1 이상 2.2 이하의 모든 실수를 저장하라!
    → ??? (실수 사이에는 무한의 수가 존재함)
  - 컴퓨터에서 실수는 항상 약간의 오차를 가지고 있음
    → 오차를 줄이려는 노력은 프로그래머가 직접함

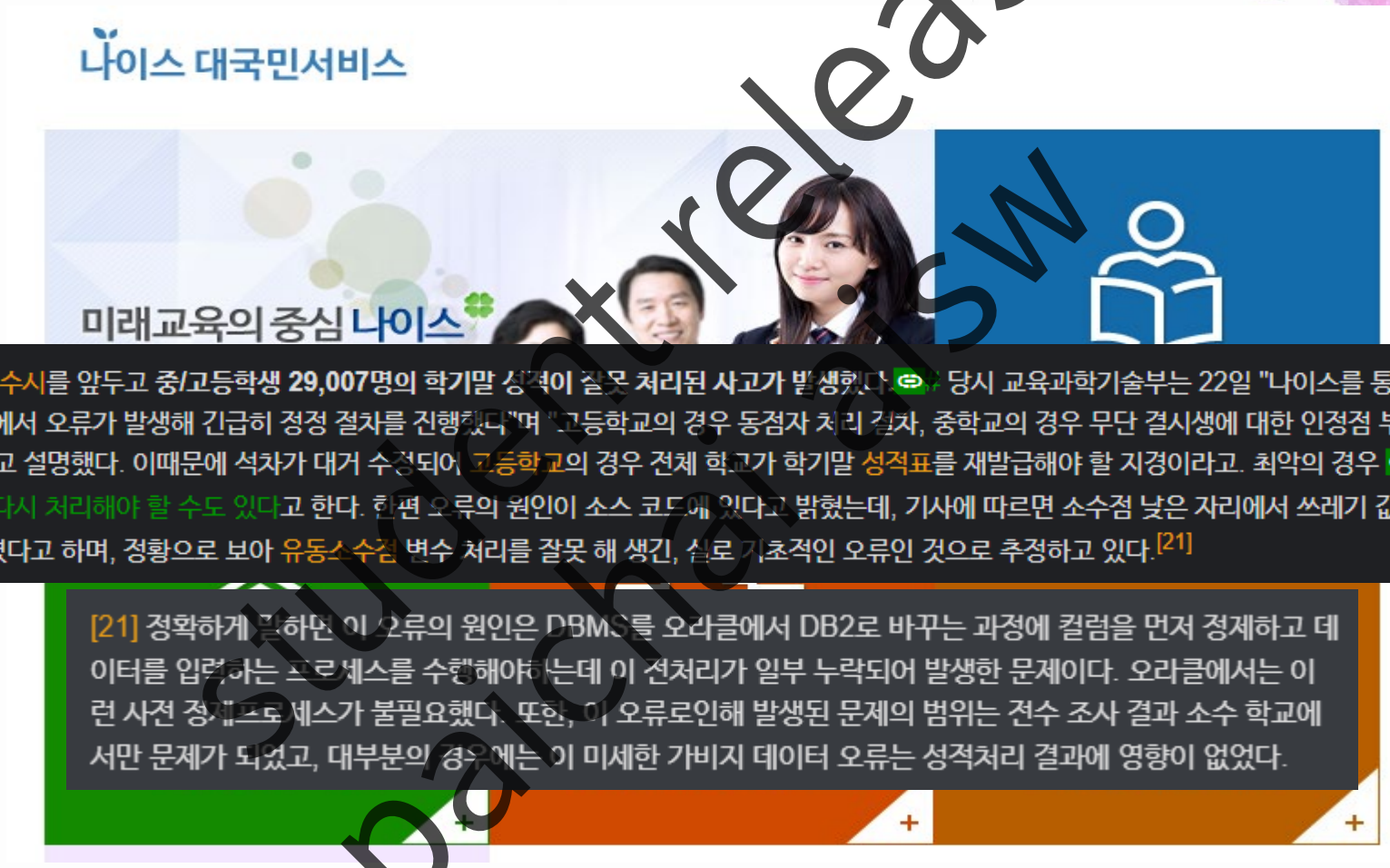
```
>>> num = 1.00000000000001    # 실수는 오차가 존재
>>> print(num)
1.00000000000001
```

  - 실수 연산이 진행될수록 오차가 눈에 더욱 띄게 됨

```
>>> num1 = 1.00000000000001
>>> num2 = 1.1
>>> print(num1 + num2)
2.1000000000000014
```

1-1. 저장 방식

- 실수 오차로 생긴 버그로 인한 치명적 문제 발생 예



- 2011년 7월, 수시를 앞두고 중/고등학생 29,007명의 학기말 성적이 잘못 처리된 사고가 발생했다. 당시 교육과학기술부는 22일 "나이스를 통한 학기말 성적 처리 과정에서 오류가 발생해 긴급히 정정 절차를 진행했다"며 "고등학교의 경우 동점자 처리 절차, 중학교의 경우 무단 결시생에 대한 인정점 부여 절차에 오류가 생겼다"고 설명했다. 이때문에 석차가 대거 수정되어 고등학교의 경우 전체 학교가 학기말 성적표를 재발급해야 할 지경이라고. 최악의 경우 중/고등학생 전체 성적을 다시 처리해야 할 수도 있다고 한다. 한편 오류의 원인이 소스 코드에 있다고 밝혔는데, 기사에 따르면 소수점 낮은 자리에서 쓰레기 값[20]에 의한 수치가 빈발하였다고 하며, 정황으로 보아 유동소수점 변수 처리를 잘못 해 생긴, 실로 기초적인 오류인 것으로 추정하고 있다.[21]

[21] 정확하게 말하면 이 오류의 원인은 DBMS를 오라클에서 DB2로 바꾸는 과정에 컬럼을 먼저 정제하고 데이터를 입력하는 프로세스를 수행해야하는데 이 전처리가 일부 누락되어 발생한 문제이다. 오라클에서는 이런 사전 정제프로세스가 불필요했다. 또한, 이 오류로인해 발생된 문제의 범위는 전수 조사 결과 소수 학교에서만 문제가 되었고, 대부분의 경우에는 이 미세한 가비지 데이터 오류는 성적처리 결과에 영향이 없었다.

1-1. 저장 방식

- 실수 오차로 생긴 버그로 인한 치명적 문제 발생 예

2002년 1월16일 미 우주왕복선 콜롬비아호가 이륙했다. 1981년 이래 28번째 우주발사였다. 이륙 직후 외부 연료탱크에서 떨어져나간 발포단열재 조각이 왕복선의 왼쪽 날개에 부딪혔다. 이미 오래 전에 `안전을 위협하지 않는 단순한 현상'이라는 평가를 받았던 사고여서 미항공우주국(NASA)의 관제본부는 그냥 넘어갔다.

1994년 인텔은 매출액 100억 달러를 넘기며 세계 최대의 컴퓨터칩 제작사로 부상했다. 주가는 연일 상승했다. 그러던 어느 날 인텔이 만든 인터넷포럼 게시판에 `펜티엄 FPU에 오류있음'이라는 한 줄의 글이 올라왔다. 인텔 개발자들은 즉각 "90억 번만에 한번씩 나눗셈에서의 근사값이 잘못 나타나는 것으로 사용자가 2만7000년만에 한번 마주칠까 말까할 정도의 칩 설계 오류"라고 일축했다.

사소한 문제가 큰 문제를 초래할 수도 있다!

1-1. 저장 방식

- 사소한 버그의 예

LoL 양 진영 대포 미니언 사거리 실험 영상 /유튜브 채널 'Vandal'

"대칭 구조의 맵에서 팀전을 치르는 경기에서 기본 세팅된 봇의 공격 범위가 알고 보니 달랐다. 이는 '버그(개발 상의 오류)'였고, 무려 10년이 넘는 기간 동안 발견되지 않았다. 그 사이 수많은 유저들은 '기울어진 운동장' 위에서 수없이 랭크 게임(점수가 매겨지는 게임)'을 치렀다. 글로벌 프로 리그 역시 마찬가지였다."

지난달 28일, 게임 역사에 길이 남을 기깃말 같은 사건이 발생했습니다. 바로 '리그 오브 레전드(League of Legends·LoL)' 블루 팀 대포 미니언 사거리가 레드 팀보다 7%(300:280)가량 길게 설정됐다는 사실이 게임 출시 10년 하고도 9개월 만에 발견된 겁니다.

이에 유저들은 즉각 대포 미니언을 활용한 각종 실험에 돌입했습니다. 한 유튜버가 실험 결과 인간 플레이어가 개입하지 않는 라인전 상황에서 블루 팀은 23번 승리했으나, 레드 팀이 이긴 건 10번뿐이었습니다. 블루 팀 승률이 2배가 넘는 겁니다.

1-2. 산술 연산자

- 정수와 실수 타입에서 사칙 연산 가능 (나눗셈, 나머지 연산 주목)

```
>>> type(3)
<class 'int'>
>>> type(3.1)
<class 'float'>
>>> type(3.0)
<class 'float'>
```

int와 float에만 주목하라!

```
>>> 3 ** 2
9
```

```
>>> 5 / 2
2.5
```

```
>>> 5 // 2    # 나눗셈의 몫을 계산
2
>>> 5 % 2     # 나눗셈의 나머지 계산
1
```

+	덧셈
-	뺄셈
*	곱셈
**	거듭제곱
/	실수형 나눗셈
//	정수형 나눗셈
%	나머지가 얼마?

1-3. float와 int

- float 함수: int형(정수)이나 문자열을 float형(실수)으로 바꿔 줌

```
>>> num = 10
>>> num = float(num)
>>> type(num)
<class 'float'>
```

```
>>> height = float(input("키 정보 입력: "))
키 정보 입력: 165.3
>>> print(height)
165.3
```

- int 함수: float형(실수)나 문자열을 int형(정수)으로 바꿔줌
  → float형(실수)이면 소수점 이하를 버림

```
>>> num = int(3.14)
>>> print(num)
3
```

```
>>> height = int(input("키 정보 cm 단위로 입력: "))
키 정보 cm 단위로 입력: 178
>>> print(height)
178
```

1-4. 복합 대입 연산자

- 대입 연산자와 산술 연산자의 결합

```
>>> num = 10
>>> num += 1 # num = num + 1을 줄인 표현
>>> print(num)
11
```

```
num = num - 1  vs. num -= 1

num = num * 3  vs. num *= 3
```

`+=`	`a = a + b`
`-=`	`a = a - b`
`*=`	`a = a * b`
`**=`	`a = a ** b`
`/=`	`a = a / b`
`//=`	`a = a // b`
`%=`	`a = a % b`

- 대입 연산자 왼편엔 저장할 수 있는 것!
  오른편엔 값이 올 수 있음!

1-5. 소괄호

- 수학에서 소괄호는 연산의 순서를 먼저 진행하라는 의미
  → 파이썬도 이와 같은 기능을 함

```
>>> 3 + 4 / 2
5.0
>>> (3 + 4) / 2
3.5
```

연습 문제

- 예제 1. 원의 반지름(radius)을 입력받아
  원의 넓이를 구하는 프로그램을 작성하시오.
  (circle.py로 저장)
- 예제 2. 체질량 지수(BMI: Body Mass Index)는
  자신의 몸무게(kg)를 키(m)의 제곱으로 표현한 것이다.
  몸무게와 키를 입력 받아
  BMI를 구하는 프로그램을 작성하시오. (bmi.py로 저장)

$$\text{신체질량지수 (BMI)} = \frac{\text{체중(kg)}}{[\text{신장(m)}]^2}$$

계산식	신체질량지수(BMI)=체중(kg)/[신장(m)]2	
판정기준	저체중	20 미만
	정상	20 - 24
	과체중	25 - 29
	비만	30 이상
장단점	표준체중보다는 체지방을 비교적 정확하게 반영할 수 있으면서도 매우 간단히 계산하여 판정할 수 있다.	

연습 문제

- 예제 3. 1평은 3.3058m²이다.
  제곱미터를 평으로 전환하려면
  제곱미터를 3.3058로 나눠주면 된다.
  제곱미터를 입력 받아 평으로 바꿔주는
  프로그램을 작성하시오. (pyeong.py로 저장)
  예시) 제곱미터 입력: *84*
  84 제곱미터는 25.409885평(25평) 입니다.
- 예제 4. cm로 표현된 키를 입력해 피트와 인치로 변환하는
  프로그램을 작성하라.
  1피트는 12인치, 1인치는 2.54cm이다. (feet.py로 저장)
  예시) 키를 입력하시오(cm): *163*
  163cm는 5피트 4.173228인치 입니다.

02

문자열과 리스트

1. print 함수
2. 리스트와 인덱싱
3. 리스트와 슬라이싱
4. 문자열과 리스트
5. 리스트 관련 함수
6. 문자열 관련 함수
7. 리스트의 값 삭제

2-1. print 함수

- print 함수의 복습과 더 배울 것
  - 앞서 사용한 print 함수:
    기본적으로 print 함수는 출력을 끝내면 줄을 바꿔 버림

```
print(1)
print(2)
print(3)
1
2
3
```

  - 줄을 바꾸지 않고 이어서 출력하도록 하는 방법: end 사용

```
print(1, end = '_')
print(2, end = '_')
print(3, end = '_')
1_2_3_
```

출력 마치고 나면 줄 바꾸는 거 대신, _을 출력하라!

```
print(1, end = ' ')
print(2, end = ' ')
print(3, end = ' ')
1 2 3
```

출력 마치고 나면 줄 바꾸는 거 대신, 빈 공간을 출력하라!

2-2. 리스트와 인덱싱

- 파이썬에서 데이터로 인식하는지 여부 확인

```
>>> 35
35
```

int형 데이터로 인식함

```
>>> Happy
Traceback (most recent call last):
  File "<pyshell#0>", line 1, in <module>
    Happy
NameError: name 'Happy' is not defined
```

Happy는 어떠한 데이터로도 인식하지 못함
문자열은 큰따옴표나 작은따옴표로 묶어야 함

```
>>> 3.14
3.14
```

float형 데이터로 인식함

2-2. 리스트와 인덱싱

- 리스트(list): 여러 개의 데이터를 묶을 때 사용하는 파이썬 데이터 타입
  - 대괄호([])로 데이터를 묶을 수 있음

```
>>> type([1, 2, 3])
<class 'list'>
```

  - 서로 다른 종류의 데이터로 얼마든지 묶을 수 있음

```
>>> [1, "hello", 3.3]
[1, 'hello', 3.3]
```

  - 리스트 안에 리스트를 넣을 수 있음

```
>>> [1, 2, [3, 4], ["AAA", "ZZZ"]]
[1, 2, [3, 4], ['AAA', 'ZZZ']]
```

  - 리스트를 변수에 담는 것도 가능

```
>>> st = [1, "hello", 3.3]
>>> print(st)
[1, 'hello', 3.3]
```

2-2. 리스트와 인덱싱

- 지금까지 배운 파이썬 데이터 타입 종류 정리
  - int형 데이터 예) 3, 5, 100
  - float형 데이터 예) 2.2 100.0 3.14159
  - 문자열 데이터 예) "I am a boy", "Girl", "Taejun"
  - **리스트 데이터 예) [1, 2, 3], ["AB", "CD], [2.5, "AB"]**

- 리스트형 데이터 연산: 합(+)과 곱(*) 연산이 가능함 (이후 인덱싱, 슬라이싱)
  - 합: 리스트를 연결하는 기능
  - 곱: 리스트를 곱해서 연결하는 기능 (반복하는 기능임)

```
>>> [1, 2, 3] + [4, 5]
[1, 2, 3, 4, 5]
```

합과 곱이 되니,
*복합 대입 연산자에서 +=와 *=도 당연히 된다.*

```
>>> [1, 2, 3] * 2
[1, 2, 3, 1, 2, 3]
```

2-2. 리스트와 인덱싱

- 인덱싱(indexing): 리스트는 여러 개의 데이터가 있으므로 한 개씩 뽑는다는 의미
  → 뽑을 때 어떤 것을 이용하는가? 숫자를 이용!
- 인덱싱 연산에서 주의: **첫번째 값을 가져올 때 1이 아니라 0**
- 인덱싱 연산이 대입 연산자(=) 오른쪽에 오면 값을 꺼낸다는 의미

```
>>> st = [1, 2, 3, 4, 5]
>>> n1 = st[0]
>>> n2 = st[4]
>>> print(n1, n2)
1 5
```

st[0]를 n1에 저장
→ 변수 st에 담긴 리스트에서 첫번째 값을 꺼내 변수 n1에 저장하라

- 인덱싱 연산이 대입 연산자(=) 왼쪽에 오면 값을 수정한다는 의미

```
>>> st = [1, 2, 3, 4, 5]
>>> st[0] = 5
>>> st[4] = 1
>>> st
[5, 2, 3, 4, 1]
```

st[0]를 5로 변경
→ 변수 st에 담긴 리스트에서 첫번째 값을 5로 수정하라

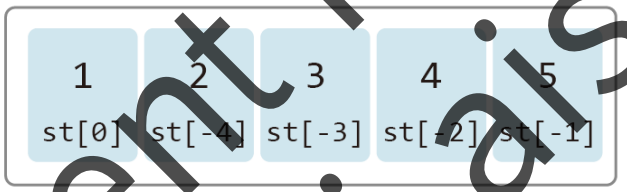
2-2. 리스트와 인덱싱

- 인덱싱 연산이 홀로(단독으로) 쓰이면 값을 꺼낸다는 의미

```
>>> st = [1, 2, 3, 4, 5]
>>> print(st[0], st[2], st[4])
1 3 5
```

print(1, 3, 5)를 의미하는 것임

- 인덱싱 연산은 음수도 가능함: 양수면 오른쪽 방향, 음수이면 반대 방향



  - 양수 인덱싱 값에 첫번째 원소는 0이고,
    음수 인덱싱 값에 마지막 원소는 -1

```
>>> st = [1, 2, 3, 4, 5]
>>> print(st[-1], st[-2], st[-3])
5 4 3
```

연습 문제

- 예제 1. 리스트를 다음과 같이 선언하여 인덱싱 연산을 이용해 리스트에 담긴 내용을 하나씩 출력해보자.
  이 때, 인덱스 값은 음수만 사용하라.
  >>> st = [1, 2, 3, 4]
- 예제 2. 리스트를 다음과 같이 선언하고 인덱스 연산을 이용해 리스트 안에 담긴 값을 모두 1씩 증가하라.
  >>> st = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
- 예제 3. 리스트를 다음과 같이 선언하고 첫번째 값과 마지막의 값을 교환하라.
  >>> st = [1, 2, 3, 4, 5, 6]

```
[Hint] 두 값의 교환
>>> x, y = 2, 7
>>> x, y = y, x   # x와 y에 저장된 값 교환
>>> print(x, y)
7 2
```

2-3. 리스트와 슬라이싱

- 인덱싱 연산과 유사함
- 인덱싱 연산: 리스트에 속한 1개의 값을 대상으로 수행하는 연산
  슬라이싱 연산: 리스트에 속한 값을 1개 이상 묶어서
  이를 대상으로 수행하는 연산

```
>>> st1 = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> st2 = st1[2:5]          # st1[2:5]를 꺼내 st2에 저장
>>> st2
[3, 4, 5]
```

st1[2]부터 st1[4]까지의 값을 꺼내 st2에 리스트 형태로 저장하라

- 슬라이싱 연산 주의: 콜론(:) 뒤에 숫자까지 연산이 아니라
  (주어진 수 – 1)까지 꺼냄
  예) st1[2:5] → st1[2]부터 st1[4]까지 꺼낸다는 의미
- 반열림구간 [a, b) 이라고 생각하면 됨

2-3. 리스트와 슬라이싱

- 인덱싱 연산과 마찬가지로 값을 꺼내고 수정이 가능함

```
>>> st = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> st[2:5] = [0, 0, 0]
>>> st
[1, 2, 0, 0, 0, 6, 7, 8, 9]
```

```
>>> st = [1, 2, 3, 4, 5, 6, 7, 8, 9]
>>> st[2:5] = [0, 0, 0, 0, 0]
>>> st
[1, 2, 0, 0, 0, 0, 0, 6, 7, 8, 9]
```

2-3. 리스트와 슬라이싱

- 콜론(시작:끝)을 기점으로 시작 생략
  → 시작이 0(리스트 첫번째 위치)이라면 생략 가능
- 콜론(시작:끝)을 기점으로 끝 생략
  → 끝까지(리스트 마지막 위치)이면 생략 가능

```
>>> st = [1, 2, 3, 4, 5]
>>> st[0:3] = [0, 0, 0]
>>> st
[0, 0, 0, 4, 5]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> st[:3] = [0, 0, 0]
>>> st
[0, 0, 0, 4, 5]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> st[2:] = [0, 0, 0]
>>> st
[1, 2, 0, 0, 0]
```

2-3. 리스트와 슬라이싱

- 콜론(시작:끝)을 기점으로 시작과 끝 모두 생략
  → 리스트 전체를 대상으로 한다는 의미

```
>>> st = [1, 2, 3, 4, 5]
>>> st[:] = [0, 0, 0, 0, 0]
>>> st
[0, 0, 0, 0, 0]
```

- 슬라이싱 연산을 이용해서 리스트를 부분적으로 교체할 수 있음
  → 교체할 값의 수가 정확히 일치하지 않아도 됨

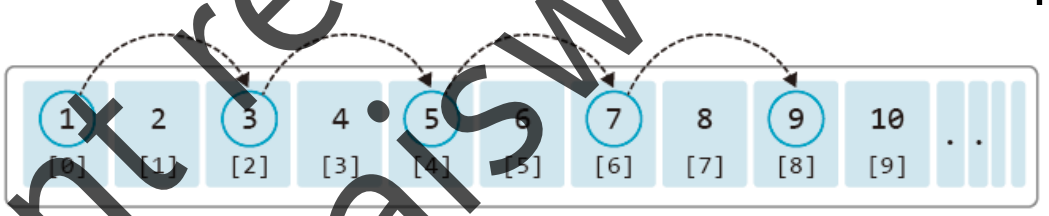
```
>>> st = [1, 2, 3, 4, 5]
>>> st[:] = [0]          # 리스트 전체를 0 하나로 교체
>>> st
[0]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> st[:] = []           # 리스트 전체 내용 삭제
>>> st
[]
```

2-3. 리스트와 슬라이싱

- n칸 씩 뛰면서(징검다리) 값을 가져오는 연산
  → 콜론(start:end:steps) 방식으로 steps를 주면 됨

```
>>> st1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> st2 = st1[0:9:2]      # st1[0] ~ st2[8]까지 2칸씩 뛰면서
>>> st2
[1, 3, 5, 7, 9]
```




```
>>> st1 = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15]
>>> st2 = st1[0:9:3]      # st1[0] ~ st2[8]까지 3칸씩 뛰면서
>>> st2
[1, 4, 7]
```

연습 문제

- 예제 4. 다음 리스트 대상으로 2와 4를 삭제하라.
  (2와 4를 삭제하지 말고, 2, 3, 4를 3 하나로 교체하라)
  >>> st = [1, 2, 3, 4, 5]
- 예제 5: 다음 리스트 대상으로 3과 4 사이에 3.5를 추가하라.
  (값을 끼워넣지 말고, 일부 내용으로 교체하라)
  >>> st = [1, 2, 3, 4, 5]
- 예제 6: 다음 리스트를 대상으로 2, 3, 4를 삭제하라.
  (빈 리스트 []로 2, 3, 4를 대체하라)
- 예제 7: 다음 리스트를 대상으로 짝수 번째 위치만 뽑아서 변수 nt에 저장하라. (인덱싱 연산을 사용하지 마라)
  >>> st = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

2-4. 문자열과 리스트

- 문자열(string): 문자들을 나열한 데이터
  → 숫자를 묶어서 문자열로 표현하면 숫자가 아닌 문자열이 됨
  예) '12345' : 이것은 문자열이 됨
- 문자열 표현:
  파이썬에서는 큰따옴표("")와 작은따옴표('')로 묶여야 함

```
>>> "Happy birthday to you"
'Happy birthday to you'
```

```
>>> 'Happy birthday to you'
'Happy birthday to you'
```

- 문자열도 파이썬 데이터 타입 중 하나임

```
>>> type('what1')      # 작은따옴표로 묶은 문자열
<class 'str'>
>>> type("what2")      # 큰따옴표로 묶은 문자열
<class 'str'>
```

2-4. 문자열과 리스트

- 지금까지 배운 파이썬 데이터 타입 종류 정리
  - int형 데이터 예) 3, 5, 100
  - float형 데이터 예) 2.2 100.0 3.14159
  - **문자열 데이터 예) "I am a boy", "Girl", "Taejun"**
  - 리스트 데이터 예) [1, 2, 3], ["AB", "CD], [2.5, "AB"]

2-4. 문자열과 리스트

- 문자열은 리스트와 유사한 성질이 있음
  → 하나 이상의 값을 묶어서(모아서) 만드는 데이터인 공통점

```
>>> [1, 2] + [3, 4]
[1, 2, 3, 4]
>>> "Hello" + "Everybody"
'HelloEverybody'
```

리스트의 합, 문자열의 합 리스트의 곱, 문자열의 곱 연산이 비슷함

*합과 곱이 되니, 복합 대입 연산자에서 +=와 *=도 당연히 된다.*

```
>>> st = [1, 2, 3, 4, 5]
>>> st[2]    # 세 번째 위치의 값만 뽑아 냄
3
>>> str = "SIMPLE"
>>> str[2]          # 세 번째 위치의 값만(문자만) 뽑아 냄
'M'
```

리스트의 인덱싱 연산, 문자열의 인덱싱 연산이 비슷함

2-4. 문자열과 리스트

- 문자열은 리스트와 유사한 성질이 있음
  → 하나 이상의 값을 묶어서(모아서) 만드는 데이터인 공통점

```
>>> st = [1, 2, 3, 4, 5, 6, 7]
>>> st[2:5]
[3, 4, 5]
>>> str = "SIMPLEST"
>>> str[2:5]
'MPL'
```

리스트의 슬라이싱 연산,
문자열의 슬라이싱 연산이 비슷함

- 문자열과 리스트이 차이점:
  리스트와 달리 문자열은 그 내용물을 바꿀 수 없다

```
>>> str = "Happy"
>>> str[0] = "D"
Traceback (most recent call last):
  File "<pyshell#1>", line 1, in <module>
    str[0] = 'D'
TypeError: 'str' object does not support item assignment
```

문자열(사람이 쓰는 말)은
한 번 내뱉으면 바꿀 수 없다는 것과 비슷

2-4. 문자열과 리스트

- 문자열과 리스트의 길이를 재는 len 함수
  → 문자열과 리스트의 길이(데이터 개수)를 반환함

```
>>> st = [1, 2, 3]
>>> len(st)
3
>>> sr = 'HaHaHa~'
>>> len(sr)
7
```

- 리스트 안에 최대, 최소값을 반환하는 max와 min 함수

```
>>> st = [2, 5, 3, 7, 4]
>>> min(st)
2
>>> max(st)
7
```

연습 문제

- 예제 8. 리스트 안에 있는 값을 거꾸로 출력하라.
  슬라이싱 연산을 잘 활용해서 해결하라.
  >>> st = [1, 2, 3, 4, 5]

- 예제 9. 문자열 안에 있는 값을 거꾸로 출력하라.
  "예제 8"과 비슷한 방식으로 활용하라.
  >>> a = "ABCDEFG"

2-5. 리스트 관련 함수

- 리스트 변수에 점(.)을 찍으면 리스트와 관련된 함수들이 나옴!
  → 관련된 함수를 보려면 Ctrl + Space를 누르면 자동으로 보여줌!
  (이러한 기능을 "자동 완성" 기능이라고 합니다!)
- remove(), append(), extend()

```
>>> st = [1, 2, 3]
>>> st.remove(2)          # 리스트에서 2를 찾아서 삭제
>>> st
[1, 3]
```

```
>>> st = [1, 2, 3]
>>> st.append(4)          # st의 끝에 4 추가
>>> st.extend([5, 6])     # st의 끝에 [5, 6]의 내용 추가
>>> st
[1, 2, 3, 4, 5, 6]
```

2-5. 리스트 관련 함수

- insert(), clear(), append()

```
>>> st = [1, 2, 4]
>>> st.insert(2, 3)     # 인덱스 값 2의 위치에 3 저장
>>> st
[1, 2, 3, 4]
>>> st.clear()      # 리스트 내용 전부 삭제
>>> st
[]
```

```
>>> st = []         # 빈 리스트 생성
>>> st.append(1)    # 리스트에 1 추가
>>> st.append(9)    # 리스트에 9 추가
>>> st
[1, 9]
```

2-5. 리스트 관련 함수

- pop(), remove(), count(), index()

```
>>> st = [1, 2, 3, 4, 5]
>>> st.pop(0)     # 인덱스 값 0의 위치에 저장된 데이터 삭제
1
>>> st
[2, 3, 4, 5]
>>> st.remove(5)     # 리스트에서 5를 삭제
>>> st
[2, 3, 4]
```

```
>>> st = [1, 2, 3, 1, 2]
>>> st.count(1)     # 1이 몇 번 등장하는지 세어라.
2
>>> st.index(2)     # 처음 2가 등장하는 위치의 인덱스 값은?
1
```

연습 문제

- 예제 10. 아래 리스트가 주어졌을 때, 값을 뒤에서부터 꺼내라.
  (힌트: pop 함수를 사용하라.)

```
>>> st = [1, 2, 3]
```

- 예제 11. 리스트를 전부 삭제하는 clear 함수를 사용하지 말고, 슬라이싱 연산으로 바꿔보자.

```
>>> st = [1, 2, 3]
>>> st.clear()     # clear 하지 말고, 슬라이싱 연산으로
>>> st
[]
```

연습 문제

- 예제 12. 하나의 리스트에 다른 리스트 값 전부를 추가해주는 extend 함수를 리스트의 합 연산, 슬라이싱 연산으로 각각 교체하라. (2개를 풀어야 하는 것임)

```
>>> st = [1, 2]
>>> st.extend([3, 4, 5])
      # 이 부분 교체(리스트의 합 연산, 슬라이싱 연산)
>>> st
[1, 2, 3, 4, 5]
```

2-6. 문자열 관련 함수

- 점을 찍고 볼 수 있는 함수들과 함수 자체를 사용하는 방식 비교
  1. 함수 자체를 사용해서 함수의 입력으로 리스트나 문자열을 주는 방식 (함수 자체를 사용하는 방식)
     예) len, max, min 함수
  2. 리스트나 문자열 변수에서 점을 찍어 함수를 사용하는 방식 (점을 찍고 볼 수 있는 함수들)
     예) st = [1, 2, 3] → st.insert(), st.pop(), st.clear() 등

  ➢ 왜 이런 방식으로 나뉘는가??
  - 첫 번째 방식은 다른 데이터 모두를 상대해야 하므로 사용하기 편해야 함
    예) len 함수는 문자열도 되고, 리스트도 상대함
  - 두 번째 방식은 한 데이터에 특화된 기능(함수)을 제공하기 위한 것임
    예) 리스트에서 insert, pop, clear 함수는 리스트만 상대함

  - 자바를 배우신 분이라면? 객체(object) 때문입니다.

2-6. 문자열 관련 함수

- count(), lower(), upper()

```
>>> str = "YoonSungWoo"
>>> str.count("o")     # "o"가 몇 번 등장?
4
>>> str.count("oo")     # "oo"가 몇 번 등장?
2
```

```
>>> org = "Yoon"
>>> lcp = org.lower()     # 모든 문자를 소문자로 바꿔서 반환
>>> ucp = org.upper()     # 모든 문자를 대문자로 바꿔서 반환
>>> org    # 원본은 그대로 존재한다.
'Yoon'
```

```
>>> lcp
'yoon'
>>> ucp
'YOON'
```

앞서 언급했듯이, 리스트와 문자열이 다른 점은 문자열은 수정할 수 없으므로 새로운 문자열이 나옴

2-6. 문자열 관련 함수

- lstrip(), rstrip(), strip()

```
>>> org = " MIDDLE "
>>> cp1 = org.lstrip()     # 앞쪽에(왼쪽에) 있는 공백들 모두 제거
>>> cp2 = org.rstrip()     # 뒤쪽에(오른쪽에) 있는 공백들 모두 제거
>>> org
' MIDDLE '
>>> cp1
'MIDDLE '
>>> cp2
' MIDDLE'
```

```
>>> org = " MIDDLE "
>>> cpy = org.strip()      # 앞과 뒤에 있는 공백들 모두 제거
>>> org
' MIDDLE '
>>> cpy
'MIDDLE'
```

앞서 언급했듯이, 리스트와 문자열이 다른 점은 문자열은 수정할 수 없으므로 새로운 문자열이 반환됨

2-6. 문자열 관련 함수

- replace(), split()

```
>>> org = "YoonSungWoo"
>>> rps = org.replace("oo", "ee")     # "oo"를 전부 "ee"로 교체
>>> rps
'YeenSungWee'
```

```
>>> org = "YoonSungWoo"
>>> rps = org.replace("oo", "ee", 1)   # 첫 번째 "oo"를 "ee"로 교체
>>> rps
'YeenSungWoo'
```

```
>>> org = "ab_cd_ef"
>>> ret = org.split('_')    # '_' 기준으로 문자열 쪼개서 리스트에 담아!
>>> ret
['ab', 'cd', 'ef']
```

2-6. 문자열 관련 함수

- find(), rfind()

```
>>> str = "What is important is that you should choose what is best for you"
>>> str.find("is")     # "is"가 있는 위치의 인덱스 값은?
5
```

```
>>> str = "What is important is that you should choose what is best for you"
>>> str.rfind("is")     # 마지막 "is"가 있는 위치의 인덱스 값은?
49
```

2-6. 문자열 관련 함수

- 문자열의 특수문자: 특별한 의미를 가지도록 약속한 문자 조합
  - 역슬래시, 원화 기호 주의 (한글/영문 글꼴에 따라 다름)

```
>>> str = "escape₩ncharacters"
>>> print(str)
escape
characters
```

₩n	줄 바꿈
₩t	탭
₩'	작은따옴표 출력
₩"	큰따옴표 출력

큰 따옴표와 작은 따옴표를 혼용해서 사용하는 방식 (문자열의 경계 구분에 전혀 상관 없음)

```
>>> str = "제가 마음속으로 그랬습니다. '이건 아니야.'라고 말이죠."
>>> print(str)
제가 마음속으로 그랬습니다. '이건 아니야.'라고 말이죠.
```

이스케이프 문자를 사용하는 방식 (문자열의 경계 구분에 전혀 상관 없음)

```
>>> str = '제가 마음속으로 그랬습니다. \'이건 아니야.\'라고 말이죠.'
>>> print(str)
제가 마음속으로 그랬습니다. '이건 아니야.'라고 말이죠.
```

연습 문제

- 예제 13. 아래 문자열을 한 번은 대문자로 모두 바꾸고, 다시 소문자로 모두 바꿔라.

```
>>> str = "The Espresso Spirit"
```

- 예제 14. 우리나라 주민등록번호에서 앞자리만 출력하라.

```
>>> p1 = "980118-1987125"
>>>  # 이 부분을 잘 생각해보세요!
980118
```

  - 참고사항: 우리나라 주민등록번호를 정수 int로 저장할까요? 아니면 문자열 str로 저장할까요? 그 이유는 무엇일까요?

2-7. 리스트의 값 삭제

- 리스트에서 값을 삭제하는 명령은 여러가지가 있음
  (문자열은 값을 수정할 수 없기 때문에 언급 안 함)

```
>>> st = [1, 2, 3, 4, 5]
>>> st.clear()       # 리스트의 모든 값 삭제
>>> st
[]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> st[:] = []       # 리스트의 모든 값 삭제
>>> st
[]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> st[2:] = []      # 인덱스 2 이후로 전부 삭제
>>> st
[1, 2]
```

2-7. 리스트의 값 삭제

- 리스트에서 값을 삭제하는 명령은 여러가지가 있음
  (문자열은 값을 수정할 수 없기 때문에 언급 안 함)

```
>>> st = [1, 2, 3, 4, 5]
>>> del st[:]          # 리스트에 저장된 값 모두 삭제
>>> st
[]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> del st[3:]         # st[3]부터 그 뒤까지 모두 삭제
>>> del st[0]          # st[0] 하나만 삭제
>>> st
[2, 3]
```

```
>>> st = [1, 2, 3, 4, 5]
>>> del st             # 리스트를 통째로 삭제! 리스트 자체를 삭제!
```

03

참과 거짓 그리고 조건문

1. 참과 거짓
2. 관계 연산자
3. 논리 연산자
4. if ~ else
5. if ~ elif ~ else
6. 참과 거짓 반환 함수
7. in과 not in

3-1. 참과 거짓

- 참과 거짓: 어떤 명제(조건)에 대한 결과

```
>>> True      # True는 그 단어의 의미처럼 '참'을 뜻한다.
True
>>> False     # False는 그 단어의 의미처럼 '거짓'을 뜻한다.
False
```

- 질문에 대한 대답이 참인지 거짓인지를 의미하는 데이터

```
>>> 3 > 10          # 3이 10보다 크니?
False
>>> 3 < 10          # 3이 10보다 작으니?
True
```

3-1. 참과 거짓

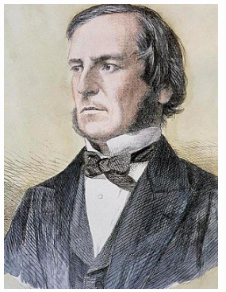
- 참과 거짓을 의미하는 데이터를 변수에 저장할 수 있음
- 변수는 데이터를 저장하기 위한 공간이며 이를 접근하고 수정할 수 있음

```
>>> r = 3 < 10      # < 연산의 결과인 True가 변수 r에 저장된다.
>>> r
True
```

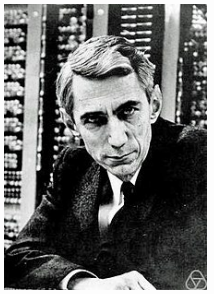
- 참과 거짓을 의미하는 데이터 타입은?

```
>>> type(True)
<class 'bool'>
>>> type(False)
<class 'bool'>
```

부울(Boolean)형 데이터 타입이라고 함



George Boole
수학/논리학자



C. E. Shannon
디지털 회로 창시
정보이론 창시

wikipedia.org

3-1. 참과 거짓

- 지금까지 우리가 배운 데이터 타입 정리
  - int형 데이터 예) 3, 5, 100
  - float형 데이터 예) 2.2 100.0 3.14159
  - 문자열 데이터 예) "I am a boy", "Girl", "Taejun"
  - 리스트 데이터 예) [1, 2, 3], ["AB", "CD], [2.5, "AB"]
  - **부울형 데이터 예) True, False**

3-2. 관계 연산자

- 참(True)과 거짓(False)를 반환하는 연산자 – 관계 연산자
- 두 수의 관계(부등호)를 의미함

A > Z	A가 Z보다 크면 True, 크지 않으면 False 반환
A < Z	A가 Z보다 작으면 True, 작지 않으면 False 반환
A >= Z	A가 Z보다 크거나 같으면 True, 그렇지 않으면 False 반환
A <= Z	A가 Z보다 작거나 같으면 True, 그렇지 않으면 False 반환
A == Z	A와 Z가 같으면 True, 같지 않으면 False 반환
A != Z	A와 Z가 같지 않으면 True, 같으면 False 반환

같음! 수학에서 흔히 말하는 '='와 다름!

다름!

위 식이 참이면 True, 아니면 False!

3-3. 논리 연산자

- 참(True)과 거짓(False)를 반환하는 연산자 – 논리 연산자
- 둘 다 만족해야 하는 AND(~이고), OR(~이거나), NOT(~이 아님)
  → 관계 연산자로 구성된 수식을 이어주는 역할

```
>>> True and True
True
>>> True and False
False
```

논리?
~이고,

신발을 사고 싶은데...
검정색이어야 하고, 가벼워야 되! 그래야 나는 사!
→ 둘 다 참을 만족해야 참!

```
>>> True or False
True
>>> False or False
False
```

~이거나,

신발을 사고 싶은데...
검정색이거나 가벼우면 되! 그러면 나는 사!
→ 둘 중에 하나가 참이면 참!

```
>>> not False
True
>>> not True
False
```

~이 아님 ...

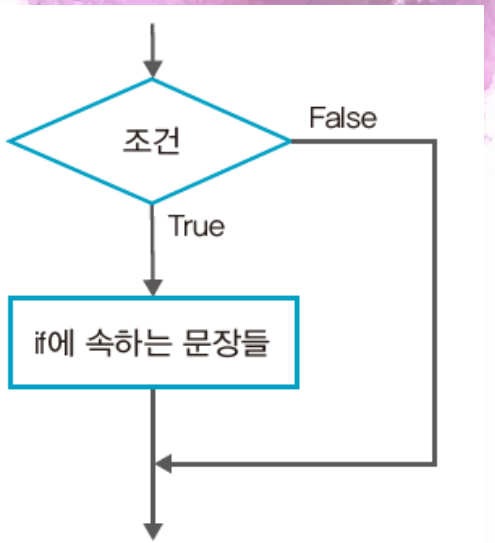
신발을 사고 싶은데...
검정색만 아니면 되! 그러면 나는 사!
→ 반대

3-4. if ~ else

- 특정한 조건이 참(True)일 때 실행하라! – if

조건이 True이면 들여쓰기된 문장이 실행된다!

```
if <조건>:
        <if에 속하는 문장1>
        <if에 속하는 문장2>
        . . . .
```



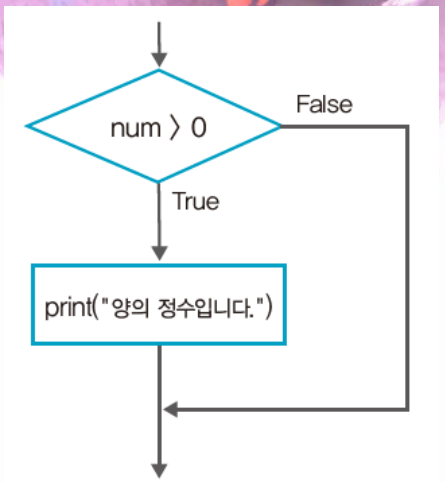
➢ 파이썬에서는 들여쓰기가 굉장히 중요한 역할임!
→ 들여쓰기를 잘못하면 에러가 발생함

여기서 말하는 조건이란! 앞서 배운 관계 연산자와 논리 연산자로 이뤄져서 결과가 True 아니면 False가 나오는 식을 말합니다!

3-4. if ~ else

- 특정한 조건이 참(True)일 때 실행하라! – if

```
if num > 0:
    print("양의 정수입니다.")
```



많은 개발자들은 main 함수를 두는 것을 더 선호함!

```
# if_positive.py
def main():             # main 함수의 정의
        num = int(input("정수 입력: "))
        if num > 0:
                print("양의 정수입니다.")

main()                  # 위의 main 함수를 실행해라!
```

정수 입력: 2

양의 정수입니다.

만약 문장이 1개라면 이렇게 써도 상관없다. 이후에도 마찬가지이다.

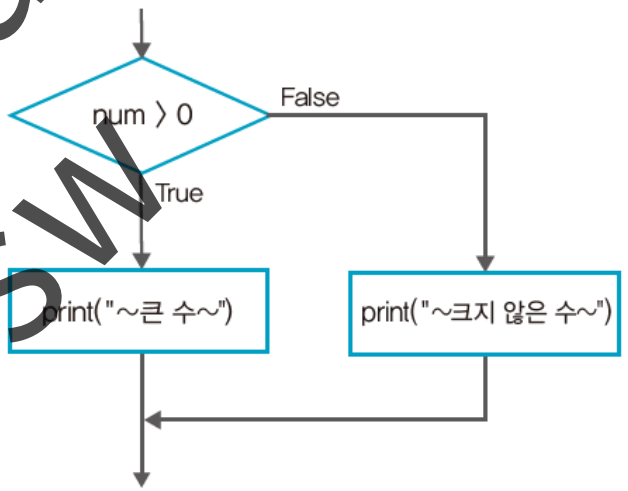
```
>>> num = 2
>>> if num > 0: print("양의 정수입니다.")
```

3-4. if ~ else

- 특정한 조건이 참(True)일 때 실행하라! - if
  참이 아니고 거짓(False)이면 이것도 실행하라! - else

```
# if_else.py
def main():   main 함수의 정의
        num = int(input("정수 입력: "))
        if num > 0:
                print("0보다 큰 수입니다.")
        else:
                print("0보다 크지 않은 수입니다.")

main()  main 함수의 호출
```



정수 입력: -7

0보다 크지 않은 수입니다.

3-5. if ~ elif ~ else

- 특정한 조건이 참(True)일 때 실행하라! – if
  참이 아니고 거짓이면 이 조건도 한 번 실행하라! – elif (else if)
  참이 아니고 거짓(False)이면 이것도 실행하라! – else
  - elif는 if와 짝을 반드시 이뤄야 함 (else와 마찬가지)
  - elif는 몇 개를 써도 상관 없음 (if와 else는 무조건 1번씩만 가능함)

```
# if_elif_else.py
def main():
        num = int(input("정수 입력: "))

        if num > 0:
                print("0보다 큰 수입니다.")
        elif num < 0:
                print("0보다 작은 수입니다.")
        else:
                print("0으로 판단이 됩니다.")

main()
```

이게 맘에 드는데 이거 사지!
엥? 없네... 그러면 저게 더 나아 보이는데?
에이 뭐 아무것도 맘에 드는 것 없네!

정수 입력: 0
0으로 판단이 됩니다.

3-5. if ~ elif ~ else

- 활용 예

관계 연산자 ==는 같냐고 물어보는 겁니다!

```
# if_elif_else2.py
def main():
        num = int(input("정수 입력: "))
        if num == 1:
                print("1을 입력했습니다.")
        elif num == 2:
                print("2를 입력했습니다.")
        elif num == 3:
                print("3을 입력했습니다.")
        else:
                print("1, 2, 3 아닌 정수 입력했습니다.")

main()
```

정수 입력: 2

2를 입력했습니다.

3-5. if ~ elif ~ else

- 활용 예

신발을 사고 싶은데...
검정색이어야 하고, 가벼워야 되! 그래야 나는 사!
→ 둘 다 참을 만족해야 참!

```
# if_and_if.py
def main():
        num = int(input("2의 배수이면서 3의 배수인 수 입력: "))
        if num % 2 == 0:
                if num % 3 == 0:
                        print("OK!")
                else:
                        print("NO!")
        else:
                print("NO!")

main()
```

```
2의 배수이면서 3의 배수인 수 입력: 6
OK!
```

```
if (num % 2) == 0 and (num % 3) == 0:
        print("OK!")
else:
        print("NO!")
```

3-5. if ~ elif ~ else

- 관계 연산자 같음(==)과 다름(!=)을 리스트와 문자열에도 활용

```
>>> 'abc' == 'abc'          # 두 문자열이 같은가?
True
>>> 'abc' != 'abc'          # 두 문자열이 다른가?
False
```

```
>>> [1, 2, 3] == [1, 2]     # 두 리스트가 같은가?
False
>>> [1, 2, 3] != [1, 2]     # 두 리스트가 다른가?
True
```

연습 문제

- 예제 1. 사용자로부터 정수를 하나 입력 받아(input 함수) 그 값에 대해 아래와 같이 한 개의 답변을 하도록 프로그램을 만들어라.
  답변) 입력한 값은 0이거나 0보다 큽니다!
  입력한 값은 0보다 작습니다!

  실행 예시 1) 값 입력: *1*
  입력한 값은 0이거나 0보다 큽니다!
  실행 예시 2) 값 입력: *-2*
  입력한 값은 0보다 작습니다!

연습 문제

- 예제 2. 파이썬에서는
  아래와 같이 관계 연산자를 이용할 수 있다.

```
>>> num = 3
>>> 1 < num < 5
True
```

이러한 방식은 다른 프로그래밍 언어에서는
지원하지 않는다. 그러면 다른 곳에서는 어떻게 쓸까?
즉, 1보다 크면서 5보다 작은 수를 어떻게 작성할까?
힌트: 관계 연산자와 논리 연산자를 활용하라.

연습 문제

- 예제 3. 사용자로부터 월을 입력 받아,
  그 값에 대해 다음 중 한 가지 답변을 하도록 코드를 만들어라.

  답변) 이번 달은 31일까지 있네요.
  이번 달은 30일까지 있네요.
  (힌트: 4월, 6월, 9월, 11월인 경우)
  이번 달은 28일까지 있네요.
  (힌트: 2월인 경우)

  실행 예 1) 월 입력: *3*
  이번 달은 31일까지 있네요.
  실행 예 2) 월 입력: *4*
  이번 달은 30일까지 있네요.

연습 문제

- 예제 4. 위 “예제 3”을 조금 변형하여 연도를 추가로 입력 받고 윤년인 경우를 생각하라.
  윤년이면, 2월은 28일이 아니라 29일까지 있다.
  → 윤년은 4로 나눠 떨어져야 하고,
  100으로 떨어지면 안된다.
  다만, 400으로 나눠 떨어지는건 된다.

실행 예 1) 연 입력: *2020*
월 입력: *2*
이번 달은 29일까지 있네요.

3-6. 참과 거짓 반환 함수

- isdigit(), isalpha()

```
>>> st1 = "123"
>>> st2 = "OneTwoThree"
>>> st1.isdigit()                    # st1은 숫자로만 이뤄져 있나요?
True
>>> st2.isdigit()                    # st2는 숫자로만 이뤄져 있나요?
False
```

```
>>> st1 = "123"
>>> st2 = "OneTwoThree"
>>> st1.isalpha()
False
>>> st2.isalpha()
True
```

3-6. 참과 거짓 반환 함수

- startswith(), endswith()

```
>>> str = "Supersprint"
>>> str.startswith("Super")      # 문자열이 'Super'로 시작하는가?
True
>>> str.endswith("int")      # 문자열이 'int'로 끝나는가?
True
```

- isdigit()와 startswith() 활용 예

```
# is_phone_num.py
def main():
        pnum = input("스마트폰 번호 입력: ")
        if pnum.isdigit() and pnum.startswith("010"):
                print("정상적인 입력입니다.")
        else:
                print("정상적이지 않은 입력입니다.")

main()
```

스마트폰 번호 입력:
01077779999
정상적인 입력입니다.

3-6. 참과 거짓 반환 함수

- 문자열 내부의 일부를 확인하는 함수

```
s.find(sub)          # 앞에서부터 sub를 찾아 인덱스 값 반환
s.rfind(sub)         # 뒤에서부터 sub를 찾아 인덱스 값 반환
s.startwith(prefix) # prefix로 시작하면 True 반환
s.endwith(suffix)   # suffix로 끝나면 True 반환
```

```
>>> s = "Tomato spaghetti"
>>> if s.find("ghe") != -1:
        print("있습니다.")
    else:
        print("없습니다.")

있습니다.
```

3-7. in과 not in

- 찾는 내용의 존재 여부를 확인함. 들어 있는가?
  - 리스트, 문자열 대상으로 사용 가능

```
>>> s = "Tomato spaghetti"
>>> if s.find("ghe") != -1:
        print("있습니다.")
    else:
        print("없습니다.")

있습니다.
```

```
>>> if "ghe" in s:
        print("있습니다.")
    else:
        print("없습니다.")

있습니다.
```

```
>>> 3 in [1, 2, 3]      # 리스트 [1, 2, 3] 안에 3이 있는가?
True
>>> 4 in [1, 2, 3]      # 리스트 [1, 2, 3] 안에 4가 있는가?
False
```

3-7. in과 not in

- 찾는 내용의 존재 여부를 확인함. 들어 있는가?
  - 리스트, 문자열 대상으로 사용 가능

```
>>> 3 not in [1, 2, 3]
False
>>> 4 not in [1, 2, 3]
True
>>> "he" not in "hello"
False
>>> "oo" not in "hello"
True
```

- True와 False가 결과인 연산자: 관계 연산, 논리 연산, in, not in

연습 문제

- 예제 5. 사용자가 정수를 입력하면,
  그 수의 거듭제곱 값을 출력하라.
  사용자가 정수가 아닌 것을 입력하면,
  "정수가 아닙니다"를 출력하라.

- 예제 6. 패스워드 "ABC258"을 입력하면
  "로그인 되었습니다"를 출력하고,
  패스워드가 틀리면
  "패스워드를 확인하세요"를 출력하라.

참고 자료

- 숫자 int형도 True나 False로 인식될 수 있음
  - 0이 오면 False로 간주
  - 0이 아니면 True로 간주
  - 이는 C언어에서 영향을 받았음

```
>>> num = 1
>>> if num:
        print("0 아닙니다.")

0 아닙니다.
```

```
>>> num = 1
>>> if num != 0:
        print("num은 0 아닙니다.")

num은 0 아닙니다.
```

참고 자료

- bool 함수를 이용해 숫자, 문자열, 리스트의 True와 False 확인

```
>>> bool(5)
True
```

```
>>> bool("what")
True
>>> bool("")
False
```

```
>>> bool([1, 2, 3])
True
>>> bool([])
False
```

04

반복문

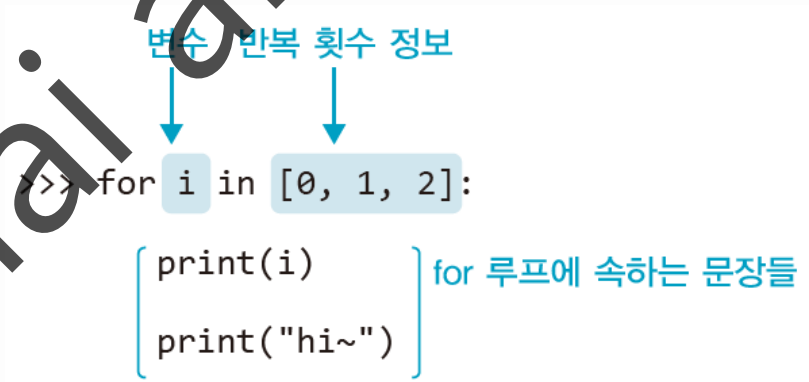
1. for 문
2. for과 range
3. while 문
4. 분기 - break
5. 분기 - continue
6. 중첩 반복문

4-1. for 문

- 루프(loop): 반복의 의미 → for 루프, while 문을 while 루프라고도 함
- for <변수> in <범위>
- 들여쓰기를 통해 for 루프에 속하는 문장들이 만들어짐
- 변수 i에 0을 넣어 for에 속한 문장들을 실행,
  변수 i에 1을 넣어 for에 속한 문장들을 실행,
  변수 i에 2를 넣어 for에 속한 문장들을 실행 → 총 3번 실행됨

```
>>> for i in [0, 1, 2]:
        print(i)
        print("hi~")

0
hi~
1
hi~
2
hi~
```



4-1. for 문

- 1 ~ 10까지 값을 더해 합을 구하는 예시

```
>>> sum = 0
>>> for i in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]:
        sum = sum + i
>>> print(sum)
55
```

연습 문제

- 예제 1. 1 ~ 10까지 값을 곱한 결과를 구하는 문제

- 예제 2. 구구단 7단을 거꾸로 출력(7 * 9 = 63부터)하는 문제

4-2. for과 range

- 1 ~ 10까지 값을 더해 합을 구하는 문제
  → [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] 이렇게 작성할 것인가?
- for과 range의 조합으로 해결 가능

```
>>> sum = 0
>>> for i in range(1, 11):
        sum = sum + i
>>> print(sum)
55
```

- range(1, 11): for의 변수 i에 1부터 11 이전의 값까지 넣어서 반복을 진행하라! (반열림구간)
- 반복의 횟수를 세는 목적으로만 사용된다면 변수를 0부터 시작하는 것이 좋음

```
>>> for i in range(0, 3):
        # 3회 반복이 목적
        print("Happy")
Happy
Happy
Happy
```

4-2. for과 range

- range 함수의 첫번째로 오는 입력을 0으로 하는 것이 관례
  - 첫번째로 오는 입력 기본값이 0으로 설정되어 있음
  - 첫번째로 오는 입력을 생략하면 자동으로 두번째 값을 따라서 0 ~ n – 1까지가 됨

```
>>> for i in range(3):    # 첫 번째 0 생략 가능
        print("Happy")
Happy
Happy
Happy
```

연습 문제

- 예제 3. for range 기반의 exp 함수 만들기 - x^y을 계산해야 함

- 예제 4. “안녕하세요”를 출력하고자 한다.
  몇 번을 출력할지 물어봐서
  “안녕하세요”를 그에 맞게 출력하시오.

  예) 안녕하세요를 몇 번 출력? 2
  안녕하세요
  안녕하세요

- 예제 5. 1부터 10까지 누적합을 구하시오.
  for문과 range를 이용해서 만드시오.
  결과 값은 55가 나와야 한다.

4-3. while 문

- 먼저 조건을 검사해 True이면 반복하고, 그 다음 또 조건을 검사함

```
for <변수> in <반복 범위>:
    <for에 속하는 문장들>


while <반복 조건>:
    <조건이 True인 경우 반복 실행할 문장들>
```

```
# for_sum_range.py
def main():
    sum = 0
    for i in range(1, 11):
        sum = sum + i
    print("sum =", sum, end = ' ')

main()
```

sum = 55

```
# while_sum.py
def main():
    i = 1
    sum = 0
    while i < 11:
        sum = sum + i
        i = i + 1
    print("sum =", sum, end = ' ')

main()
```

sum = 55

4-3. while 문

- 반복의 횟수가 정해져 있지 않을 때 사용하면 좋음

```
# while_over100.py
def main():
    i = 1
    sum = 0
    while sum <= 100:
        sum = sum + i
        i = i + 1
    print(i-1, "더했을 때의 합", sum, end = ' ')

main()
```

```
14 더했을 때의 합 105
```

연습 문제

- 예제 6. 0부터 시작해서 하나씩 증가시키면서
  9까지 출력을 보이는 코드를
  while 루프를 기반으로 작성하라.

- 예제 7. 9에서부터 값을 하나씩 감소시키면서
  0까지 출력을 보이는 코드를
  while 루프를 기반으로 작성하라.

- 예제 8. 다음 수식의 빈 칸에 들어갈 수를 찾는 코드를 작성하라.
  빈 칸에 들어갈 수는 1부터 시작해서 1씩 증가시켜
  가면서 찾아라.
  다음 수식에서 빈 칸이 42이라면 정답으로 인정한다.
  3 * (빈 칸) / 2 = 63

4-4. 분기 - break

- 반복의 분기(branch): break
  - while이나 for 루프에서 사용하는 분기
  - break를 만나면 루프를 빠져나감 (반복문이 멈춰짐)

```
# while_break.py
def main():
    i = 0
    while i < 100:
        print(i, end = ' ')
        i = i + 1
        if i == 20:
            break

main()
```

i가 20이면 while 루프를 빠져 나가라!
break가 if문 안에 있어도 if문 자체가 while 루프 안에 있다.

0 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19

4-4. 분기 - break

- 무한 루프(반복이 무한히 지속됨)와 찰떡 궁합

```
# while_over100_break.py
def main():
    i = 1
    sum = 0
    while True:
        sum = sum + i
        if sum > 100:
            print(i, "더했을 때의 합", sum, end = ' ')
            break
        i = i + 1

main()
```

while True이면 반복 조건이 항상 True이기 때문에 빠져 나갈 수 없는 루프 형성이를 무한루프라고 함 (조심해서 써야 함!)

무한루프에는 반드시 조건으로 루프를 빠져나갈 수 있어야 함

14 더했을 때의 합 105

연습 문제

- 예제 9. 6과 45의 최소공배수를 구하는 코드를 while 루프 기반으로 작성하라.
  6과 45의 최소공배수는
  6으로도 나눠지고 45로도 나눠지는 값들 중
  가장 작은 값을 의미한다.
  따라서, 45부터 시작해 값을 1씩 증가시켜 가면서
  6과 45로 나눠지는 첫 번째 값을 찾으면 된다.
  코드는 아래를 참고해서 작성하라.

```
lcm = 0    # 변수 lcm을 선언해, 필요하면 추가로 변수를 선언해도 됨
           # 변수 lcm에 최소 공배수를 찾아 저장하는 코드를 구성하라.
           # 변수 lcm에 저장된 값을 출력하라. 90이 나오면 정답이다.
```

연습 문제

- 예제 10. 42와 120의 최대공약수를 구하는 코드를 while 루프 기반으로 작성하라.
  42와 120의 최대공약수는 42로도 나눌 수 있고 120으로도 나눠지는 값들 중 제일 큰 값을 의미한다.
  따라서, 42부터 시작해서 값을 1씩 감소시켜 가면서 42와 120을 나눌 수 있는 값을 찾으면 된다.
  코드는 "예제 9"의 변수 lcm을 gcm으로 변경해서 작성하고, 6이 나오면 정답이다.

4-5. 분기 - continue

- 반복의 분기(branch): continue
  - while 루프나 for 루프에서 사용하는 분기문
  - continue를 만나면 반복문을 종료하지 않고, 조건을 검사해 반복을 계속 이어가는 역할
    → continue 밑에는 실행을 건너뛰는 것임


```
>>> for i in range(1, 11):
        print(i, end = ' ')

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

```
>>> for i in range(1, 11):
        if i % 2 == 0:
            continue
        print(i, end = ' ')

1 3 5 7 9
```

짝수이면 반복의 처음으로 돌아가라!
나머지 아래에 있는 문장은 건너 뛰어라!

4-5. 분기 - continue

- 반복의 분기(branch): continue

```
>>> i = 0
>>> while i < 10:
        i = i + 1
        print(i, end = ' ')

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

```
>>> i = 0
>>> while i < 10:
        i = i + 1
        if i % 3 == 0: continue
        print(i, end = ' ')

1 2 4 5 7 8 10
```

3의 배수이면 반복의 처음으로 돌아가라!
나머지 아래에 있는 문장은 건너 뛰어라!

연습 문제

- 예제 11. 구구단에서 7단을 출력하라.
  단, 값이 홀수인 경우에만 출력하라.
  continue를 이용하라.
  (즉, 7은 출력하고 14는 출력되지 말아야 한다.)

- 예제 12. 2 이상 100 미만의 정수 중 2로도 나눠지지 않고
  동시에 3으로 나눠지지 않는 수를 찾아 출력하라.
  이를, while 루프 기반으로 작성하라.

- 예제 13. "예제 12"에서 continue를 사용하지 않았다면
  사용하는 방식으로 바꿔라.
  만약, continue를 사용했다면
  사용하지 않는 방식으로 바꿔라.

4-6. 중첩 반복문

- 이중 루프(중첩 반복문)

```
>>> for i in [1, 2]:      바깥쪽 for루프
        for j in ['a', 'b', 'c']:      안쪽 for루프
            print(j * i, end = ' ')

a b c aa bb cc
```

```
>>> sr = ['father', 'mother', 'brother']
>>> cnt = 0
>>> for s in sr:
        for c in s:
            if c == 'r':      r이 단어에서 몇 개 있는가?
                cnt += 1

>>> cnt
4
```

연습 문제

- 예제 14. 이중 for 루프를 기반으로
  구구단 2단 ~ 9단까지 전부 출력하라.

  예시) 2 4 6 8 10 12 14 16 18
  3 6 9 12 15 18 21 24 27
  4 8 12 16 20 24 28 32 36
  …
  9 18 27 36 45 54 63 72 81

- 예제 15. 이중 for 루프를 기반으로 아래와 같이 별을 출력하라.

```
******
*****
****
***
**
*
```

실전 문제

- 각 문제를 소스 코드 형태로
  파일명(solution1.py, solution2.py, …)로 저장하라.

- 문제 1. 예제에서 못 풀었던 문제 풀기
- 문제 2. 피보나치 수열은 아래와 같이 정의된다.

$$f_0 = 0$$
$$f_1 = 1$$
$$f_{i+1} = f_i + f_{i-1} \ (for\ i = 1, 2, \ldots)$$

피보나치 수열은 앞의 2개의 원소를 합해서
뒤의 원소를 만든다.
피보나치 수열은 컴퓨터에서 탐색 문제 등에서
사용되기도 한다. 아래와 같이 결과가 나오도록 하시오.

몇 번째 항까지? *10*
0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55,

실전 문제

- 문제 3. 2와 100 사이에 모든 소수(prime number)를 찾는 프로그램을 작성하라.
  소수는 1과 자기 자신만을 약수로 가져야 한다.

- 문제 4. 중첩 반복문을 사용해 아래와 같이 출력하라.
  1
  1 2
  1 2 3
  1 2 3 4
  1 2 3 4 5
- 문제 5. 서로 다른 n개에서 r개를 택해 일렬로 나열하는 수를 순열(permutation)이라 하고, nPr이라고 한다.

$$n(n-1)(n-2)\dots(n-r+1)$$

  n과 r은 사용자가 입력하게 하라.
  예시) n = 10, r = 3일 때, 순열 값은 720이다.

다음 시간

1. 튜플과 레인지
2. 함수
3. 모듈
4. 딕셔너리

참고 자료

- 윤성우, 열혈 파이썬(기본편/중급편), 오렌지미디어, 2017.
- 히로세 츠요시, 파이썬으로 배우는 게임 개발 (입문편/실전편), 제이펍, 2020.
- 폴 데이텔, 하비 데이텔, 안진섭, 프로그래머를 위한 Python, 성안당, 2021.
- 루시아누 하말류, 전문가를 위한 파이썬, O'REILLY, 2016.
- 브렛 슬라킨, Effective Python, 2nd edition, 길벗, 2020.