

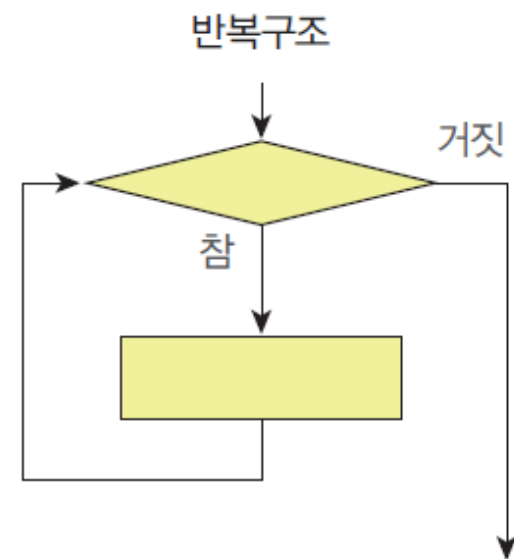
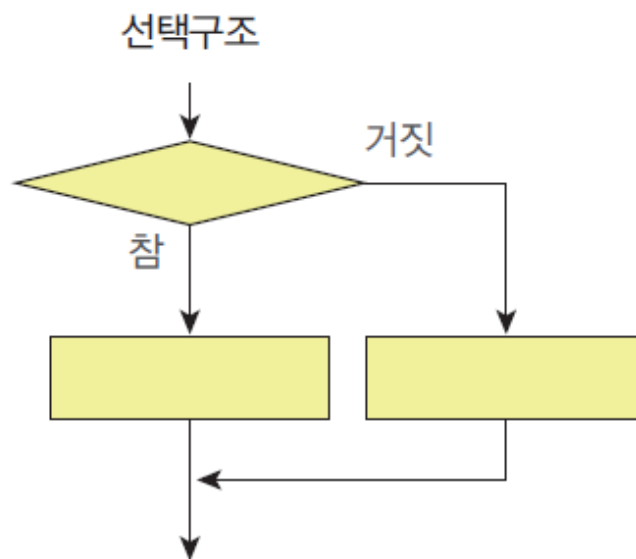
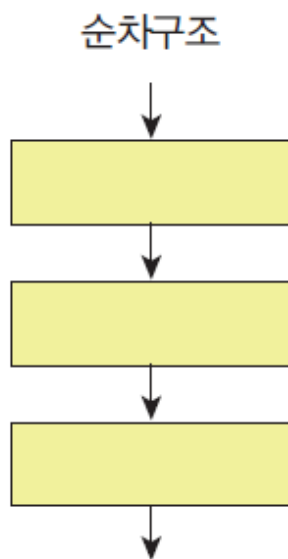


4. 파이썬 프로그래밍 기초2

- 반복문, 리스트 -

프로그래밍의 3가지 기본 제어 구조

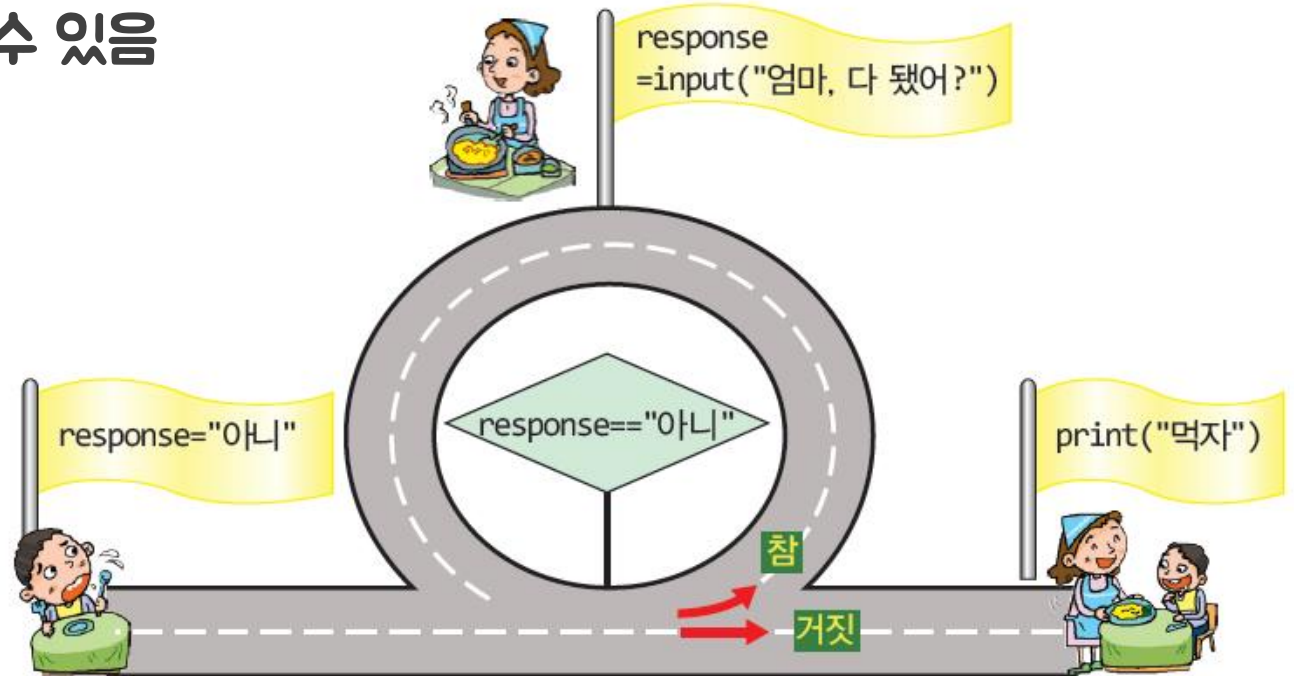
- 순차 구조: 명령들이 순차적으로 실행되는 구조이다.
- 선택 구조: 둘 중의 하나의 명령을 선택하여 실행되는 구조이다.
- 반복 구조: 동일한 명령이 반복되면서 실행되는 구조이다.



반복문

○ 어떤 조건이 만족되는 동안 반복하는 구조

⊙ 사람보다 빠르고 정확하게 반복할 수 있음



대표적인 반복문: while

○ 어떤 조건이 만족되는 동안 반복하는 구조

while 루프

while 조건 :

반복 문장

반복을 하는 조건이다. 조건이 참이면 반복을 계속한다.

반복되는 문장이다.

```
i = 0  
  
while i < 5:  
    print("i = ", i)  
    i = i + 1
```

```
i = 0  
i = 1  
i = 2  
i = 3  
i = 4
```

1부터 10까지 합을 구하는 예제

- 1부터 10까지의 모든 수를 합한다.
- 1부터 1,000,000 까지 모든 수를 합한다면 ?

```
count = 1
SUM = 0
while count <= 10:
    SUM = SUM + count
    count = count + 1

print("합계는", SUM)
```

합계는 55

패스워드 확인 예제

○ 사용자가 입력한 암호가 맞는지 체크

암호를 입력하시오: idontknow
암호를 입력하시오: 12345678
암호를 입력하시오: password
암호를 입력하시오: pythonisfun
로그인 성공

```
password = input("암호를 입력하시오: ")  
while password != "pythonisfun":  
    password = input("암호를 입력하시오: ")  
print("로그인 성공")
```

짝수와 홀수 판별 예제

- 사용자가 입력한 정수가 홀수 인지 짝수 인지를 판별, 단 음수가 입력 되면 프로그램을 종료하라.

```
# 무한 반복 (조건이 항상 참)
while True:
    n = int( input("정수를 입력하세요: ") )
    if n < 0:
        print("종료")
        break
    elif (n%2) == 0:
        print("짝수 입니다")
    else
        print("홀수 입니다")
```

반복문의 중첩

- 조건문과 마찬가지로 반복 문도 중첩 활용 가능
 - ◎ 하나의 반복 문으로 해결되지 않는 문제는 중첩된 반복 문을 사용
- 선택문을 통해 break 명령을 사용하면 가장 가까운 하나의 반복문을 중지

```
while True :  
    count = 1  
    SUM = 0  
    X = int( input("정수를 입력하세요: ") )  
    while count <= X :  
        SUM = SUM + count  
        count = count + 1  
    print("합계는", SUM)  
    retry = input("다시 할까요? (y/n) ")  
  
    if retry == 'n'  
        break
```


반복문 리뷰

- 많은 문제들이 반복적인 계산, 처리를 요구함.
- 컴퓨터는 사람보다 단순 반복 하는 일을 정확하고 신속하게 잘 할 수 있음.
 - ⊙ $1+2$ 는 사람도 컴퓨터 만큼 빨리 할 수 있지만,
 - ⊙ $1+2+3+...+N$ 처럼 N 이 커지면 커질수록 사람에 비해 컴퓨터가 빠르고 정확하게 계산함.
- 풀어야 할 문제의 성격에 따라 반복 문의 조건을 어떻게 설정할 지가 중요함.
- 문제에 따라 반복 문의 중첩 횟수를 결정하고 설계함.
- 반복 문 내부에서 선택문 & break 를 적시 적소에 사용할 수 있음.

The background features a light blue gradient with darker blue wavy horizontal bands. Three spheres are scattered across the scene: a large light blue sphere in the bottom left, a medium green sphere in the top right, and another large light blue sphere in the top right corner.

자료 구조 (Data Structure)

- 리스트 (List) -

자료 구조란?

- 문제 해결을 위해 많은 데이터를 효율적으로 저장하고, 처리하고, 관리할 수 있도록 자료들 사이의 관계를 나타내는 것.
 - ⊙ 가령 단일 값을 저장하는 개별 변수만을 ($\alpha_1, \alpha_2, \alpha_3 \dots \alpha_{100}$) 사용하면, 합을 구하거나 평균을 구하기 위해 모든 개별 변수 이름을 나열해야 함.
- 많은 양의 데이터를 효율적으로 저장 및 처리 하려면 그에 적합한 자료 구조 설계 및 활용 능력이 필요함.

파이썬에서 제공하는 기본 자료 구조

○ 리스트(List)

- 여러 가지 원소를 하나의 묶음으로 표현할 수 있는 자료형

○ 튜플(Tuple)

- 리스트와 비슷하나, 저장된 원소를 변경 또는 삭제할 수 없음

○ 집합(Set)

- 수학에서의 집합 개념의 자료형
- 원소의 중복이 허용되지 않으며, 원소들 간의 순서가 없음

○ 사전(Dictionary)

- 키와 값의 쌍으로 구성된 원소를 표현할 수 있는 자료형

리스트

- 일련의 여러 값들을 유연하게 다루는 자료구조
 - ⊙ 인덱스를 사용하여 개별 요소 접근 가능
 - ⊙ 데이터 추가, 삭제가 용이 (전체 크기/길이도 자유롭게 변경 가능)
 - ⊙ 리스트 각 요소 데이터 타입이 다를 수 있도록 허용
 - ⊙ 정렬, 검색, 수정과 같은 편리한 기능을 함께 제공

리스트의 인덱스(Index)

- 리스트 변수 선언 시 대괄호 [] 안에 들어가는 각 요소는 쉼표를 통해 구분
- 리스트의 각 요소들은 인덱스를 통해 접근 가능
- 리스트 인덱스는 0 부터 시작



`season = ['spring', 'summer', 'fall', 'winter']`

0 1 2 3

```
print(season[0], season[1], season[2], season[3])
```

리스트 데이터 타입이 제공하는 메소드

메소드	기능 설명
<code>append</code>	리스트 끝에 새로운 요소 (item) 추가
<code>insert</code>	주어진 인덱스 위치에 새로운 요소 추가
<code>remove</code>	특정 값과 동일한 요소 하나를 리스트에서 제거
<code>pop</code>	리스트 마지막 요소 하나를 제거하면서 해당 값을 가져옴
<code>index</code>	특정 값이 리스트에 저장된 인덱스 정보를 가져옴
<code>count</code>	특정 값이 리스트에 몇 개 있는지 카운트 하여 가져옴
<code>sort</code>	리스트 안의 값들을 기준으로 정렬함
<code>reverse</code>	리스트 안의 저장된 순서를 거꾸로 저장함

<https://docs.python.org/3/tutorial/datastructures.html>

리스트 메소드 활용 예시

```
>>> fruits = ['orange', 'apple', 'pear', 'banana', 'kiwi', 'apple', 'banana']
>>> fruits.count('apple')
2
>>> fruits.index('banana')
3
>>> fruits.reverse()
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange']
>>> fruits.append('grape')
>>> fruits
['banana', 'apple', 'kiwi', 'banana', 'pear', 'apple', 'orange', 'grape']
>>> fruits.sort()
>>> fruits
['apple', 'apple', 'banana', 'banana', 'grape', 'kiwi', 'orange', 'pear']
```


항목 추가: append 와 insert 차이

- append 는 리스트 뒤에 추가됨
- insert 는 지정한 인덱스 위치에 삽입 가능
(인덱스를 설정하지 않으면 리스트 앞에 추가됨)

```
>>> print(heroes)
['아이언맨', '닥터 스트레인지', '헐크']

>>> heroes.append("스파이더맨")
>>> print(heroes)
['아이언맨', '닥터 스트레인지', '헐크', '스파이더맨']

>>> heroes.insert(1, "배트맨")
>>> print(heroes)
['아이언맨', '배트맨', '닥터 스트레인지', '헐크', '스파이더맨']
>>>
```

항목 삭제: remove vs. del

○ remove() 함수는 특정 값을 가지는 원소를 삭제

```
heroes = [ "아이언맨", "토르", "헐크", "스파이더맨" ]  
heroes.remove("스파이더맨")  
print(heroes)
```

```
['아이언맨', '토르', '헐크']
```

○ del 명령어는 특정 인덱스의 원소를 삭제한다.

```
heroes = [ "아이언맨", "토르", "헐크", "스파이더맨" ]  
del heroes[0]  
print(heroes)
```

```
['토르', '헐크', '스파이더맨']
```

sort 명령어로 정렬하기

○ 리스트 원소들을 정렬

```
nums = [ 4, 1, 3, 7, 6, 5, 2 ]  
nums.sort()  
print(nums)
```

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

```
heroes = [ "아이언맨", "토르", "헐크", "스파이더맨" ]  
heroes.sort()  
print(heroes)
```

```
['스파이더맨', '아이언맨', '토르', '헐크']
```

for 반복 문 + 리스트

○ 리스트 요소들을 쉽게 접근할 수 있는 for 반복 문

```
fruits = ['orange', 'apple', 'banana', 'kiwi']  
i = 0  
while i < len(fruits):  
    print( fruits[i] )  
    i = i + 1
```

vs.

```
for item in fruits:  
    print( item )
```

orange
apple
banana
kiwi

for 반복 문에서 range() 사용

○ N 번 반복하고 싶다면?

```
N = 5
for i in range( N ):
    print(i)
```

0
1
2
3
4

○ 1 부터 N 까지 더하고 싶다면?

```
N = 10
SUM = 0
for i in range( 1, N+1 ):
    SUM = SUM + i
print(SUM)
```

55

for 문에서의 인덱스 사용 여부

- 아래 두 for 반복 문은 동일한 결과를 출력함. 무슨 차이가 있을까?

```
scores = [80, 90, 70, 100, 50]
```

```
for value in scores:  
    print(value)
```

```
for index in range( len(scores) ) :  
    print(scores[index])
```

리스트에서 각 요소 값을 하나씩
가져와 **value**에 복사함.

range로 인덱스를 자동 생성하고,
인덱스로 **리스트** 각 요소에 접근함.

예제: 리스트 요소 값 수정하기

- 만약 오답이 있어서 모두 5 점씩 올려야 한다면?

```
SUM = 0
scores = [95, 75, 85, 65]

print(scores)

for value in scores:
    value = value + 5
    print( value )

print(scores)
```

5 점씩 더해 준 결과가 리스트에 반영 되었을까?

```
SUM = 0
scores = [95, 75, 85, 65]

print(scores)

for i in range(len(scores)):
    scores[i] = scores[i] + 5
    print( scores[i] )

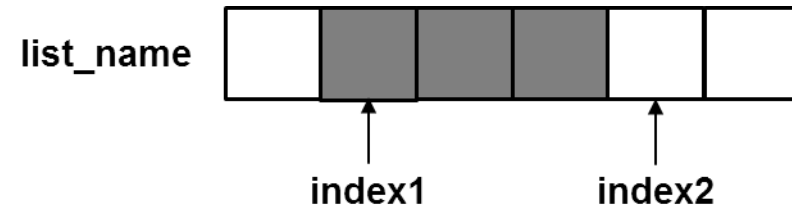
print(scores)
```

인덱스를 사용하여 리스트 요소에 직접 저장 필요!

리스트 슬라이스 (slice)

○ 리스트의 일부분을 잘라서 사용할 수 있음

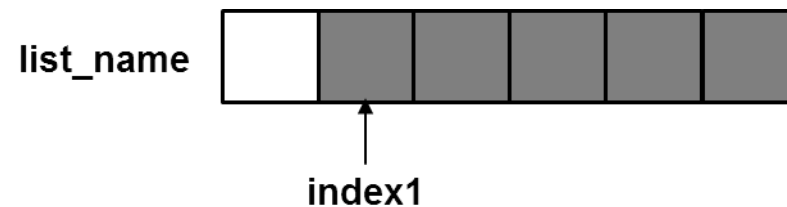
```
list_name[ index1 : index2 ]
```



```
list_name[ : index2 ]
```



```
list_name[ index1 : ]
```



예제: 리스트 슬라이스

○ BTS 멤버 중 일부를 유닛으로 구성하려고 한다.

⊙ 다음과 같이 구성할 경우 해당 멤버는 어떻게 되는가?

```
BTS = ['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']
```

```
print(BTS)
print(BTS[:3])
print(BTS[2:5])
print(BTS[4:])
```

```
['RM', '진', '슈가', '제이홉', '지민', '뷔', '정국']
```

```
['RM', '진', '슈가']
```

```
['슈가', '제이홉', '지민']
```

```
['지민', '뷔', '정국']
```

리스트와 함께 사용하기 유용한 함수들

○ `len()`

리스트의 길이를 반환

○ `max()`

리스트 요소 중 가장 큰 값을 반환

○ `min()`

리스트 요소 중 최소 값을 반환

○ `sum()`

리스트 요소 총 합을 반환

```
scores = [94, 67, 79, 80, 55]
print( len(scores) )
print( max(scores) )
print( min(scores) )
print( sum(scores) )
print( sum(scores) / len(scores) )
```

```
5
94
55
375
75.0
```

예제: 성적 관리 프로그램

- 리스트를 이용하여 N 명의 학생 수를 입력 받고,
각 학생 점수를 입력 받은 후, 모든 학생의 총합, 평균,
그리고 성적 순으로 정렬 (내림차순).

학생 수를 입력하세요: 3

성적을 입력하세요: 90

성적을 입력하세요: 100

성적을 입력하세요: 85

총합: 275 평균: 91.66666666666667

[100, 90, 85]

예제: 성적 관리 프로그램

```
st_num = int( input("학생 수를 입력하세요: ") )
scores = []

for i in range(st_num):

    print("총합: ", total_score, "평균: ", avg_score)

print(scores)          # 내림차순으로 정렬된 리스트 출력
```

