CHAPTER 8

제어문

Control Structures

박 진 수 교수
서울대학교•경영대학
jinsoo@snu.ac.kr

학습 목차

- ☑ 비교 연산자 규칙
- ☑ 조건문
  - if문
- ☑ 순환문
  - while문
  - for문
  - 무한 루프와 break/continue

비교 연산자 규칙

Comparison Operators

비교 대상

잘못된 비교 시 예외 발생

비교가 불가능한 두 개의 자료형을 비교할 경우 예외가 발생

```
'1' > 5
```

어떻게 하면 예외 없이 비교할 수 있을까?

```
int('1') > 5
```

연쇄 연결

비교 대상이 어떤 범위 안에 존재하는지 확인할 때, 프로그래밍 언어 대부분은 논리 연산자 and(또는 &)를 통해 비교 대상을 각각 비교해야 한다

```
(1 < 2) and (2 < 3)
```

```
True
```

파이썬은 비교 연산자를 여러 개 연결해서 사용할 수 있다

연쇄 연결(chaining)

```
1 < 2 < 3
```

```
True
```

표현식 하나에 여러 개의 비교 연산자를 연결해서 사용

시퀀스 자료형 비교

문자열, 튜플, 리스트와 같은 시퀀스형을 비교하는 규칙

1) 비교 연산자는 각 시퀀스형의 첫 번째 객체부터 순차로 비교하며,
2) 첫 번째 객체가 서로 같으면 그 다음 객체를 비교하고,
3) 그 다음 객체도 같으면 그 다음 객체를 비교해서,
4) 서로 다른 객체를 발견할 때까지 순서대로 비교한다.
5) 만약 서로 다른 객체를 발견하면 비교를 멈추고 그 결과를 반환한다.
6) 그 이후에 있는 객체들은 비교하지 않는다.

```
'abc' > 'abca'
```

```
('나', '다', '라', '가', '마') >= ('나', '다', '라')
```

```
('가', '나', '다', '나', '프') > ('가', '너', '다', '나', '프')
```

```
('아', '야', '어', '여', '오') <= ('아', '야', '어', '나', '흐')
```

```
('ㄱ', 'ㄴ', 'ㄷ', 'ㄹ', 'ㅁ') < ('ㄱ', 'ㄴ', 'ㄷ', 'ㅁ', 'ㄹ' )
```

```
[1, 2, 3, 4, 99] > [1, 2, 3, 5]
```

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

```
[1, 2, 3, 4, 5] <= [1, 2, 3, 2, 99]
```

```
[1, 2, 3, 4, 5] < [1, 2, 3, 99, 2]
```

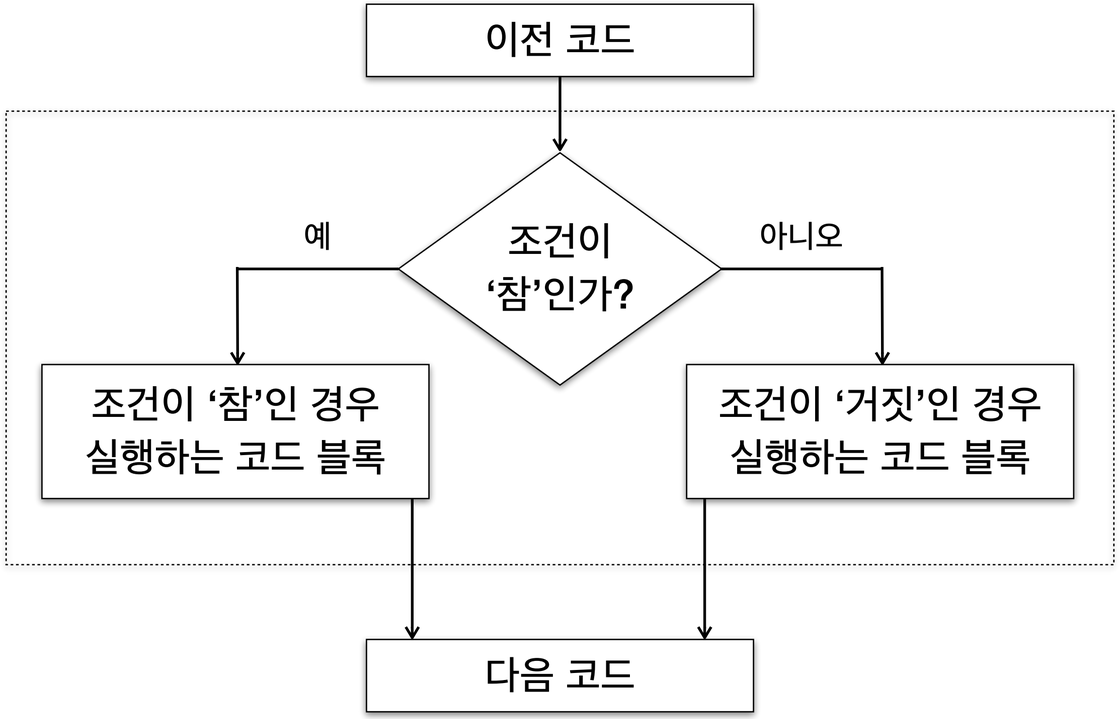
```
['apple', 'banana', 'pineapple'] <= ['apple', 'Blueberry', 'plum']
```

조건문

Conditional Branching

조건문

- **if** 문
  - 조건문을 사용하면 조건의 ‘참’과 ‘거짓’ 여부에 따라 무슨 명령문을 실행할지 결정할 수 있다
  - 주로 두 개 또는 그 이상의 대안이 주어지고 컴퓨터는 조건문의 결과에 따라 이 중 한 개의 명령문을 선택해서 실행



조건문 기본 형식

```
if 불린-표현식:
    if-명령문
```

```
if 불린-표현식:
    if-명령문
else:
    else-명령문
```

```
i = input('숫자(정수)를 입력하세요: ')
if int(i) > 0:
    print('입력한 숫자는 양의 정수입니다.')
```

```
숫자(정수)를 입력하세요:  5
입력한 숫자는 양의 정수입니다.
```

```
i = int(input('숫자(정수)를 입력하세요: '))
if i > 0:
    print('입력한 숫자는 양의 정수입니다.')
else:
    print('입력한 숫자는 양의 정수가 아닙니다.')
```

```
숫자(정수)를 입력하세요:  0
입력한 숫자는 양의 정수가 아닙니다.
```

조건문 일반 형식

조건문 작성 방법

`elif`(else if) 문은 선택 사항

`else` 문도 선택 사항이며 항상 마지막에 온다

```
if 불린-표현식-1:
  if-명령문
elif 불린-표현식-2:
  elif-명령문-1
...
elif 불린-표현식-N:
  elif-명령문-N
else:
  else-명령문
```

- `if` 문은 반드시 있어야 하지만, `elif` 문과 `else` 문 모두 선택 사항이라 생략할 수 있다
- `elif` 문의 개수는 제한이 없기 때문에 여러 개 작성할 수 있다
- `else` 문은 하나만 올 수 있고 항상 마지막에 와야 한다
- `if` 문, `elif` 문, `else` 문의 마지막에는 항상 쌍점(:)이 온다
- `불린-표현식`인 `불린-표현식-1`, `...`, `불린-표현식-N` 의 값은 '참(`True`)'과 '거짓(`False`)' 둘 중 하나만 갖는다
- 조건문은 위에서부터 아래로 순서로 실행되면서 `불린-표현식`의 값을 평가한다
  - `불린-표현식`의 값이 '참'이면 해당 명령문(`명령문-1`, `...`, `명령문-N`)을 실행한다
  - `불린-표현식` 모두가 '거짓'이면 마지막에 있는 `else-명령문`을 실행한다

예시 : 조건문

```
prompt = '정수를 입력하세요: '
result = '둘 중 더 큰 수는 {}입니다.'

x = int(input(prompt))
y = int(input(prompt))

if x > y:
    print(result.format(x))
elif x < y:
    print(result.format(y))
else:
    print('같은 숫자군요.')
```

문자열 서식설정 방법은 5장 <기본자료형> 중 문자열의 서식 설정과 <부록 4> 참고

중첩 조건문

중첩 조건문(nested if-statement)

if 문은 중첩이 가능

```
prompt = '정수를 입력하세요: '
result = '둘 중 더 큰 수는 {}입니다.'

x = int(input(prompt))
y = int(input(prompt))

if x == y:
    print('같은 숫자군요.')
else:
    if x > y:
        print(result.format(x))
    else:
        print(result.format(y))
```

간편 조건문

간편 조건문(조건식) 작성 방법

```
표현식-1 if 불린-표현식 else 표현식-2
```

```
if score > 60:
    grade = 'P'
else:
    grade = 'F'
```

=

```
if score > 60: grade = 'P'
else: grade = 'F'
```

=

```
grade = 'P' if score > 60 else 'F'
```

예시 : 간편 조건문

대체필드1

대체필드2

```
for count in range(5):
    print(f"{count if count != 0 else 'No'} file{'s' if count != 1 else ''}")
```

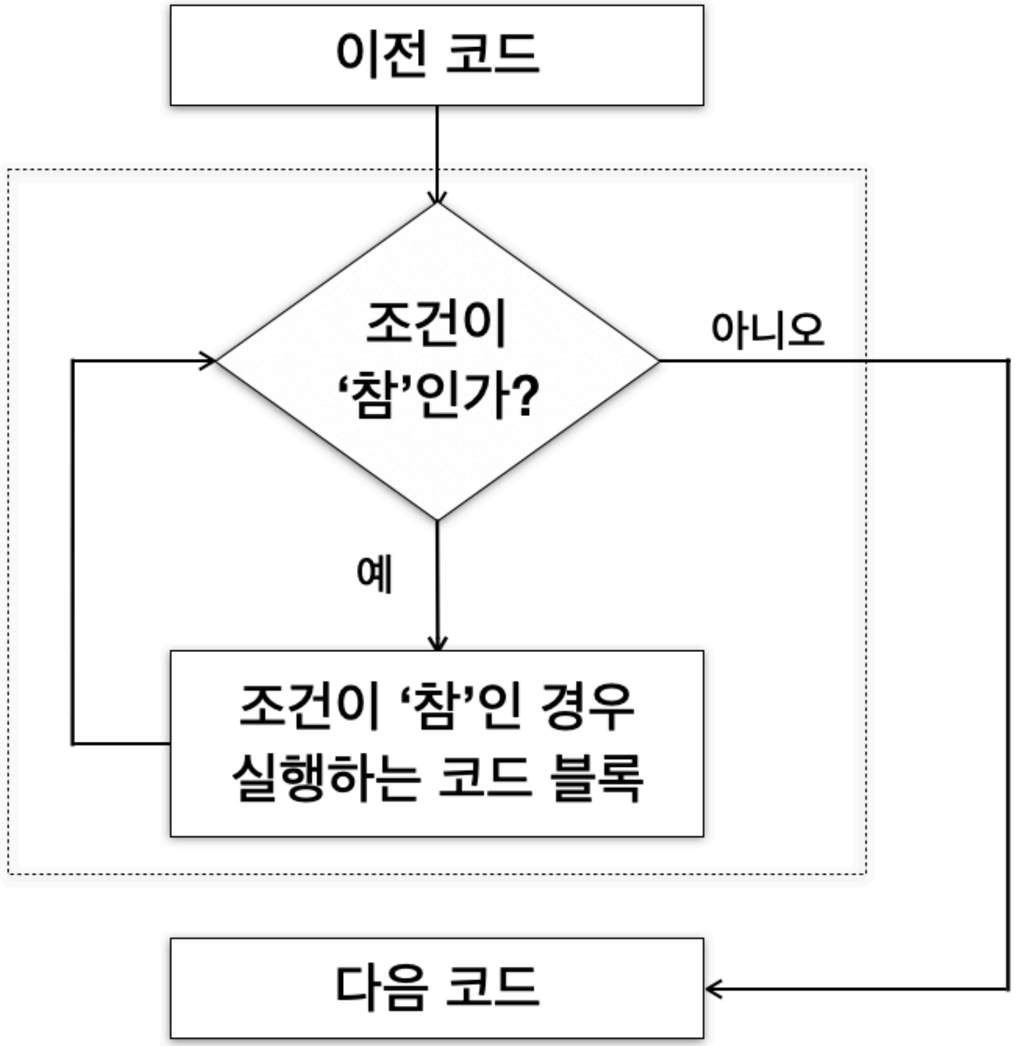
문자열 서식설정 방법은 5장 <기본자료형> 중 문자열의 서식 설정과 <부록 4> 참고

순환문

Looping

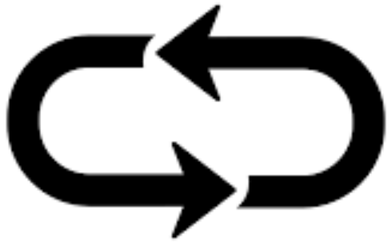
순환문

- 순환문(loop statement)이란?
  - 특정 조건에 도달 할 때까지 계속 반복해서 실행하는 일련의 명령어
    - ‘반복문’ 이라고도 한다
  - 반복 처리를 수행하는 명령문으로 `while` 문과 `for` 문이 있다
    - `while` 문
      - 주어진 조건이 '참'인 동안 `while` 문에 속한 명령문을 반복적으로 실행
    - `for` 문
      - 주어진 조건이 '참'인 동안 `for` 문에 속한 명령문을 반복적으로 실행



`while` 문

`while` statement



while 문 기본 형식

while 불린-표현식:
**   while-명령문**

```
i = 0
while i < 5:
    print('안녕 파이썬')
    i += 1
```

```
i = 0
while i < 10:
    print(i)
    i += 1
```

```
t = tuple('abcdefg')
print(t)
```

```
i = 0
while i < len(t):
    print(t[i])
    i += 1
```

while 문 일반 형식

else 문은 선택 사항

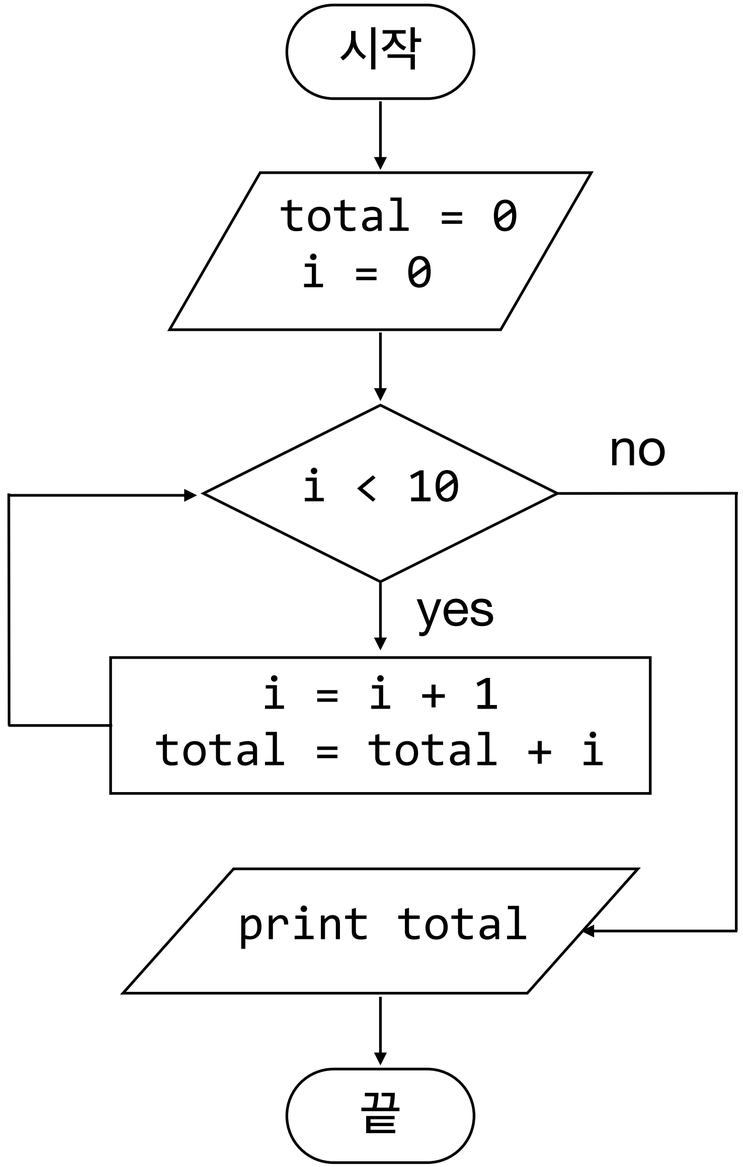
```
while 불린-표현식:
    while-명령문
else:
    else-명령문
```

while 문 작성 방법

- while 문에 있는 불린-표현식이 '참(True)'이면, while-명령문을 실행
  - 불린-표현식이 '거짓(False)'일 때까지 while-명령문을 반복해서 실행
- while 문에 있는 불린-표현식이 '거짓(False)'이면, while 문을 종료하고 else 문의 else-명령문을 실행
  - 즉, while 문을 정상적으로 종료했다면 else 문은 반드시 실행된다
- 다음 두 가지 경우에는 else 문이 실행되지 않는다
  - while 문이 break 또는 return에 의해 종료
  - while 문 실행 중 예외(exception)가 발생
- else 문 실행 규칙은 아래 명령문에 모두 적용
  - while 문
  - for 문
  - try-except 문
- else 문은 선택 사항
- while 과 else 의 마지막에는 쌍점(:)이 온다

흐름도(flowchart)

1부터 10까지 더하기



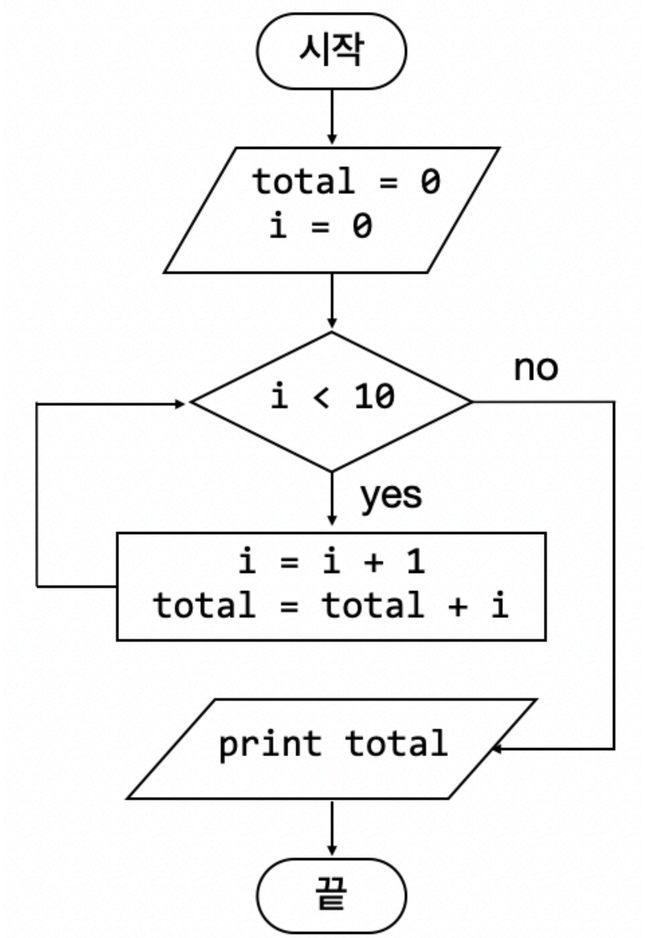

```
total = 0
i = 0

while i < 10:
    i += 1
    total += i
else:
    print(total)
```

```
55
```

의사코드

코드의 흐름이 간단하다면...

복잡한 내용이 아니라면 흐름도보다는 의사코드가 더 간단하고 편리할 수 있다




```
total = 0
i = 0

while (i < 10)
  i = i + 1
  total = total + 1
end while

print total
```

의사코드 작성법

의사(擬似)코드(슈도코드, pseudocode) 작성법

1) 정해진 규칙은 없지만
2) 최소한 프로그래밍 언어가 일반적으로 사용하는 키워드로 표현하고
3) 블록은 들여쓰기를 할 것을 권장

```
if (condition)
    statements...
else-if (condition)
    statements...
else
    statements...
end if
```

```
while (condition)
   statements...
   break if (condition)
   statements...
end while
```

```
for (condition)
   statements...
   break if (condition)
   statements...
end for
```

```
function name(input1, input2...)
   statements...
return (output1, output2...)
```

```
class name(input1, input2...)
   statements...
return (output1, output2...)
```

```
output1, output2 = call name(input1, input2...)
```

예시 : while 문

```
total = 0
i = 0
while i < 10:  # 명령문을 10회 반복해서 수행한 후 종료
    i += 1
    total += i
else:          # 결과를 출력
    print(f'1부터 10까지의 합은 {total}입니다.')
```

```
integers = 2, 8, 3, 5, 10, 27
total = 0
index = 0
while index < len(integers):  # 튜플의 모든 항목을 처리한 후 종료
    total += integers[index]  # 튜플 인덱스 0부터 len(integers) - 1, 즉 전체 객체의 합을 구한다
    index += 1                # 튜플 인덱스 번호를 1 증가시킨다
else:                         # 결과를 출력한다.
    print(f'튜플에 들어있는 숫자의 합은 {total}입니다.')
```

예시 : while 문

```
menu = '''1. 학생 추가
2. 학생 삭제
3. 학생 목록 보기
4. 종료'''

number = ''
while number != '4':
    print(menu)
    number = input('메뉴 번호를 입력하세요: ')
    if number in '1234':
        print(f'{number}번 메뉴를 선택했습니다.\n')
    else:
        print(f'{number}번은 없는 메뉴 번호입니다.\n')
else:
    print('Bye~~')
```

예시 : while 문

```
import random
fruits = ['사과', '딸기', '바나나', '블루베리', '포도']
fruit = ''
while fruit != '블루베리':
    fruit = random.choice(fruits)
    print(fruit)
else:
    print('블루베리가 선택되어 프로그램을 종료합니다.')
```

```
import random
fruits = ['사과', '딸기', '바나나', '블루베리', '포도']
fruit = ''
while fruit != '블루베리':
    fruit = random.choice(fruits)
    print(fruit)
    if fruit == '블루베리':
        break
else:
    print('블루베리가 선택되어 프로그램을 종료합니다.')
```

for 문

For statement

for 문 기본 형식

```
for 변수 in 순회형:
    for-명령문
```

```
for i in ['a', 'b', 'c', 'd', 'e']:
    print(i)
```

range 클래스

- 범위 안의 정수를 담은 순회자(iterator)를 반환
- range(끝번호)
  - 하나의 정수 전달인자 끝번호를 지정하면, 0부터 끝번호-1까지의 정수를 반환, 즉 0, 1, 2, 3, ..., 끝번호-1을 반환
- range(시작번호, 끝번호)
  - 두 개의 정수 전달인자 시작번호, 끝번호를 지정하면, 시작번호부터 끝번호-1까지의 정수를 반환, 즉 시작번호, 시작번호+1, ..., 끝번호-1을 반환
- range(시작번호, 끝번호, 폭)
  - 세 개의 정수 전달인자 시작번호, 끝번호, 폭을 지정하면, 시작번호부터 끝번호-1까지 폭만큼 건너 뛰면서 정수를 반환, 즉 시작번호, 시작번호+폭, ..., 끝번호-1을 반환

- 다음과 같은 경우 많이 사용
  - 정수의 리스트 또는 튜플을 생성할 경우
  - for 순환문에서 순회할 횟수를 지정할 경우

예시 : range 클래스

```
range(10)
```

```
list(range(10))
```

```
list(range(1, 10))
```

```
list(range(1, 10, 2))
```

```
list(range(9, 0, -1))
```

```
tuple(range(15, -21, -5))
```

예시 : range 클래스

for 문과 range 클래스

for 문에서 순회형 값을 처리할 때 range 클래스를 사용하면 코드가 훨씬 간결해진다

```
for i in range(5):
    print('안녕 파이썬')
```

for 문 일반 형식

for 문 작성 방법

else 문은 선택 사항

```
for 변수 in 순회형:
    for-명령문
else:
    else-명령문
```

- 순회형에 들어 있는 객체들을 처음부터 마지막까지 추출해서 차례로 변수에 할당한 후 for-명령문을 실행
- 변수는 순회형에서 가져온 객체를 참조하는 하나의 변수일 수도 있고, 복합 변수일 수도 있다
  - 복합 변수면 주로 튜플을 사용
- for 문을 종료한 후, else 문의 else-명령문을 실행
  - 즉, for 문을 정상적으로 종료했다면 else 문은 반드시 실행된다
  - 결국 전체 for 문은 else-명령문을 실행한 후에 종료
- 다음 두 가지 경우에는 else 문이 실행되지 않는다
  - for 문을 break 또는 return으로 종료
  - for 문 실행 중 예외(exception)가 발생
- else 문은 선택 사항
- for 와 else 의 마지막에는 쌍점(:)이 온다

예시 : for-else 문

```
total = 0
for i in range(1, 11):
    total += i
else:
    print(f'1부터 10까지의 합은 {total}입니다.')
```

```
integers = 2, 8, 3, 5, 10, 27
total = 0
for item in integers:
    total += item
else:
    print(f'튜플에 들어있는 숫자의 합은 {total}입니다.')
```

enumerate 함수

enumerate(순회형[, start=0])

- 순회형 자료를 전달인자로 받아, 순회형이 담은 개별 객체와 각 객체의 순서(인덱스 번호)를 (순번, 객체) 형식의 쌍으로 순회할 수 있는 열거형(enumerate) 객체를 반환
- *start*는 첫 번째 객체의 시작 번호
  - 선택해서 사용할 수 있는 *start*의 기본값은 0이지만 다른 값으로 대체할 수 있다

```
L = ['a', 'b', 'c']
for i, k in enumerate(L):
    print(i, k)
```

예시 : enumerate 함수

```
items = tuple('abcdefg')
print(items)
```

```
x = enumerate(items)
type(x)
```

```
for i in enumerate(items):
    print(i)
```

```
for index, item in enumerate(items):
    print(index, ':', item)
```

```
for index, item in enumerate(items, start=11):
    print(index, item)
```

```
L = list(enumerate(items, 101))
print(L)
```

```
d = dict(enumerate(items, 101))
print(d)
```

for 문과 튜플 할당(언패킹)

```
display_stand = [
    ('사과', 21),
    ('바나나', 27),
    ('블루베리', 35),
    ('딸기', 29),
    ('포도', 38)
]
```

```
for i in display_stand:
    print(i)
```

```
for fruit, number in display_stand:
    print('진열대 번호:', number, '==> 과일명:', fruit)
```

```
k = [('a', 1), ('b', 2), ('c', 3), ('d', 4), ('e', 5)]

for x, y in k:
    if y % 2 == 1:  # 번호가 홀수인 튜플의
        print(x)    # 첫 번째 객체를 출력한다.
```

for 문과 딕셔너리

```
koreng = {
    '사과': 'apple',
    '블루베리': 'blueberry',
    '딸기': 'strawberry',
    '키위':'kiwi',
    '바나나': 'banana',
    '포도': 'grape',
    '자두': 'plum'
}
```

```
for i in koreng:
    print(i)
```

```
for i in koreng.items():
    print(i)
```

```
for k, v in koreng.items():
    print(k, ':', v)
```

zip 함수

zip(*순회형)

- 두 개 이상의 **순회형** 객체를 받아 각 **순회형**의 개별 객체가 튜플인 순회자를 반환
- **i** 번째 튜플은 각 **순회형** 전달인자의 **i** 번째 객체를 담고 있다
- **순회형** 앞의 * 는 **0** 이상 불특정 다수의 위치 전달인자를 받아서 처리하는 위치 전달인자 패킹 연산자(10장에서 자세히 다룬다)
- 전달인자가 하나면, 한 개의 객체만 있는 튜플 순회자를 반환
- 만약 전달인자가 없으면(**0**개면), 빈 순회자를 반환
- **순회형** 전달인자 중 길이가 가장 짧은 전달인자의 객체를 모두 사용하면 순회자는 멈춘다(즉, 가장 적은 개수를 담은 **순회형**을 기준으로 묶음 처리를 한다)

```
x = ('a', 'b', 'c')          # 튜플(길이=3)
y = ['i', 'j', 'k', 'l']     # 리스트(길이=4)
z = '12345'                  # 문자열(길이=5)
```

```
k = zip(x, y, z)
type(k)
print(k)
```

```
list(zip(x, y, z))
```

```
tuple(zip(z, y))
```

예시 : zip 함수

```
x = ('a', 'b', 'c')          # 튜플(길이=3)
y = ['i', 'j', 'k', 'l']     # 리스트(길이=4)
z = '12345'                  # 문자열(길이=5)
```

```
for t in zip(x, y, z):
    print(t)
```

```
type(zip(x, y, z))
```

```
type(t)
```

시퀀스형 언패킹 연산자 * 를 사용하면 묶은 것(zip)을 반대로 풀(unzip) 수도 있다

```
k = [('a', 1), ('b', 2), ('c', 3)]
x, y = zip(*k)  # 시퀀스형 전달인자 언패킹

print(x)
print(y)
```

무한 루프와 break/continue

Infinite Loop & break/continue

무한루프

- 무한루프(infinite loop)란?
  - 프로그램이 끝없이 동작하는 것
  - 일반적인 무한루프란 컴퓨터에서 프로그램이 끝없이 동작하는 것으로, 루프문에 종료 조건이 없거나 종료 조건과 만날 수 없을 때 발생
- `while` 문 무한루프
  - 아래의 예에서 `while` 문 내에 종료 조건이 없으므로 `while` 문은 종료되지 않고 계속해서 '안녕 파이썬'을 출력

```
while True:
    print('안녕 파이썬')
```

- 무한루프 종료 : `ctrl+c`
  - 위의 `while` 문을 실행시킨 후에 빠져나오려면 `ctrl+c`를 눌러 종료해야 한다
  - 하지만 위와 같은 무한루프는 프로그래밍을 할 때 자주 사용하지 않으며, 주로 종료 조건을 줘서 원할 때에 무한루프를 빠져 나오게 한다

무한루프 종료하기

```
status = True
stamina = 10
while status:
    print('운동을 시킵시다. 체력이 1 감소합니다.')
    stamina -= 1
    if stamina == 0:
        status = False
else:
    print('체력이 고갈되었습니다. 더 이상 운동을 할 수 없습니다.')
```

위의 `while` 문을 실행하면 처음에는 일반적인 무한루프처럼 계속 돌아가지만 10회 후에 멈춘다

`while` 문이 한 번 돌아갈 때 마다 `stamina`가 `1`씩 감소해 `0`이 되는 순간 `status`가 `False`가 되어 무한루프를 종료하게 된다

무한루프 종료하기

break 명령어 활용

앞의 코드는 아래와 같이 break 를 활용해 작성 가능

```
stamina = 10
while True:
    print('운동을 시킵시다. 체력이 1 감소합니다.')
    stamina -= 1
    if not stamina:
        print('체력이 고갈되었습니다. 더 이상 운동을 할 수 없습니다.')
        break
```

if 문에서 stamina가 0이 아닌 이상 참(True)이므로 계속 돌아간다

하지만 stamina가 0이 되는 순간 False가 되어 while 문을 종료하게 되다

break와 continue 명령어

- 순환문의 흐름을 제어하는데 사용
  - **break**
    - break 이후의 모든 코드를 건너뛰고 break가 속한 순환문을 종료하고 빠져나온다
    - 중첩 순환문 안에 있으면 break가 속한 가장 안쪽 순환문을 종료하고 빠져나온다
  - **continue**
    - continue 이후의 모든 코드를 건너뛰고 순환문의 첫 명령문으로 되돌아가 다음 반복을 진행한다

```
while/for 순환문:
  명령문-A
  if 조건문:
    ...
    break
    ...
  명령문-B
명령문-C
```

```
while/for 순환문:
  명령문-A
  if 조건문:
    ...
    continue
    ...
  명령문-B
명령문-C
```

예시 : break 명령어

break를 이용해 while 문을 빠져나가보자

```
L = list(range(10))
print(L)

index = 0
while True:
    print(L[index])
    index += 1
    if index == 5:
        break
else:
    print('종료')
```

예시 : break 명령어

특정 값을 입력하면 무한 루프를 빠져나간다

```
while True:
    answer = input('아무거나 입력하시오(마치려면 -1을 입력): ')
    if answer == '-1':  # answer가 '-1'이면 while문을 빠져나간다
        print('종료합니다...')
        break
    print(answer)
```

<enter> 키를 누르면 무한 루프를 빠져나간다

```
while True:
    answer = input('무언가를 입력하세요(마치려면 <ENTER> 키를 누르세요): ')
    if not answer:
        print('종료합니다...')
        break
    print(answer)
```

예시 : break 명령어

```python
import random

while True:
    i = random.randint(0, 10)
    if i > 9:
        print(f'{i}: 무한 루프를 빠져나갑니다.')
        break
    else:
        print(f'{i}: 무한 루프에 빠져있습니다.')
```

예시 : continue 명령어

continue 명령어로 1~100 사이 정수 중 홀수만 더한다

```
total = 0
for i in range(1, 101):
    if i % 2 == 0:  # 짝수면 건너뛴다
        continue
    total += i
else:
    print(total)
```

예시 : break와 continue 명령어

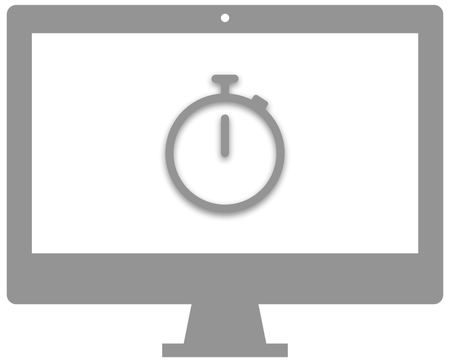
홀수만 더해서 합하고, 짝수면 건너뛰고, 음수면 즉시 for 문을 종료

```
data = [1, 4, 7, 3, 8, 6, 9, 5, 0, 2, 7, -1, 22, 31]
total = 0
for i in data:
    if i < 0:
        print(f'음수(i = {i})라 break문을 실행합니다.')
        break
    elif i % 2 == 0:
        print(f'짝수(i = {i})라 continue문을 실행합니다.')
        continue
    else:
        print(i)
        total += i
print('-' * 35)
print('홀수의 합:', total)
```

제어문

Lab Exercises

조건문 기본 형식



Lab : 단순 if 문

- 튜플이 가지고 있는 객체를 검색하기
  - 아래와 같은 튜플을 생성하여 변수 t에 할당한 후 t를 출력
    - `t = (1, 3, 5, 7, 'a', 'b', 'c')`
  - 5가 t에 있으면 '5 is in t' 출력
  - 'd'가 t에 있으면 'd is in t' 출력

  - 실행 결과 예시

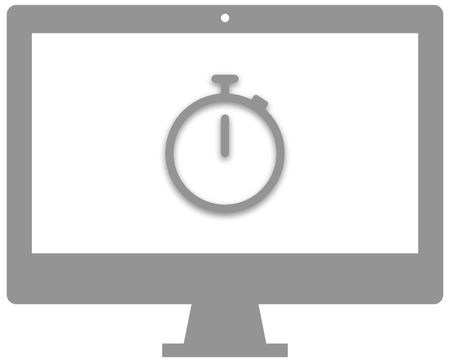
```
(1, 3, 5, 7, 'a', 'b', 'c')
5 is in t
```

Lab : if-else 문

- if-else 문 작성하기
  - input 함수를 사용해 사용자로부터 어떤 값을 입력 받아
  - 입력 값이 'on'이면 숫자 1을 출력하고 그 이외의 경우에는 항상 숫자 0을 출력하는 코드를 작성
  - 실행 결과 예시

```
조건(on/off)을 입력하세요...: on
1

조건(on/off)을 입력하세요...: off
0

조건(on/off)을 입력하세요...: 59
0
```

조건문 일반 형식



Lab : if-elif-else 문

- if-elif-else 문 작성하기
  - input 함수를 사용해 사용자로부터 어떤 값을 입력받아
  - 입력 값이 'on'이면 숫자 1을, 'off'면 -1을, 그 이외에는 항상 0을 출력하는 코드를 작성
  - 실행 결과 예시

```
조건(on/off)을 입력하세요...: on
1

조건(on/off)을 입력하세요...: off
-1

조건(on/off)을 입력하세요...: hi
0

조건(on/off)을 입력하세요...: 파이썬
0
```

Lab : 두 정수 비교

- 두 정수 비교하기
  - `input` 함수를 사용해 사용자로부터 두 정수 값을 입력받아 *x*와 *y* 변수에 할당
  - 두 변수를 비교
    - 만약 *x*가 *y*보다 크면 `'x가 y보다 큽니다'` 출력
    - 만약 *x*와 *y*가 같다면 `'x와 y가 같습니다'` 출력
    - 만약 *x*가 *y*보다 작다면 `'x가 y보다 작습니다'` 출력
  - 실행 결과 예시

```
x를 입력하세요:  7
y를 입력하세요:  5
x가 y보다 큽니다.

x를 입력하세요:  -11
y를 입력하세요:  2
x가 y보다 작습니다.

x를 입력하세요:  9
y를 입력하세요:  9
x와 y가 같습니다.
```

Lab : 다수의 대안이 주어진 조건문

- 다수의 대안이 주어진 조건문 작성하기
  - `input` 함수를 사용해 사용자로부터 색깔과 관련된 영어 단어를 입력받아 한글로 번역하는 조건문을 작성
  - 이 조건문은 다음 영어 단어에 대한 한글 번역만 가능
    - `red` –> 빨강, `blue` –> 파랑, `green` –> 초록, `white` –> 하양, `black` –> 검정
  - 그 이외, 즉 위 다섯 색깔에 포함되지 않으면 `'번역할 수 없습니다.'`를 출력
  - 조건문을 빠져나온 후 `'조건문을 빠져나왔습니다.'`를 출력
  - 실행 결과 예시

```
Choose a color: red
빨강
조건문을 빠져나왔습니다.

Choose a color: green
초록
조건문을 빠져나왔습니다.

Choose a color: gray
번역할 수 없습니다.
조건문을 빠져나왔습니다.
```

Lab : 여러 가지 조건식 작성

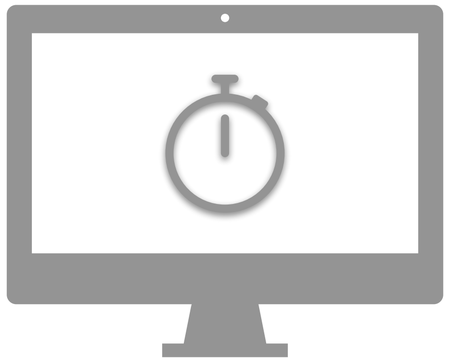
- 여러 가지 조건식 작성하기
  - 다음과 같은 리스트를 생성한 후 출력
    - `['math', 'sports', 'english', 'science', 'economics']`
  - 그리고 다음 조건을 만족하는 코드를 작성하고 결과를 출력
    - (조건 1) 배우는 과목의 숫자가 `4`개 이하일 경우 `'fail'`, `5`개일 경우 `'pass'`, 그리고 `5`보다 크면 `'liar'`를 출력
    - (조건 2) `science`와 `math`가 모두 `lectures`에 존재하면 `'이과'`, 그렇지 않으면 `'문과'`를 출력
    - (조건 3) `sports` 또는 `music` 중 하나 이상이 `lectures`에 존재하면 `'예체능'`, 그렇지 않으면 `'일반'`을 출력
  - 실행 결과 예시

```
> python multiconditions.py
['math', 'sports', 'english', 'science', 'economics']
pass
이과
예체능
```

Note: 조건식 표현법

- ***a***와 ***b***가 같다 : *a* **==** *b*
- ***a***가 ***b***보다 크거나(작거나) 같다 : *a* **>=(<=)** *b*
- ***x***가 ***a***에 포함된다 : *x* **in** *a*
- ***a*** 또는 ***b*** : *a* **or** *b*
- ***a*** 그리고 ***b*** : *a* **and** *b*

중첩 조건문



Lab : 나머지 구하기

- 나머지를 구해 그 결과를 출력하기
  - `input` 함수를 사용해 사용자로부터 두 정수 값을 입력받아 *x*와 *y* 변수에 할당
  - *x*를 *y*로 나눈 후
    - 만약 나머지가 `0`이면 `'zero'` 출력
    - 만약 나머지가 `0`이 아니고
      - 짝수면 `'even'` 출력
      - 홀수면 `'odd'` 출력
  - 실행 결과 예시

```
x를 입력하세요:  20
y를 입력하세요:  5
zero

x를 입력하세요:  20
y를 입력하세요:  7
even

x를 입력하세요:  15
y를 입력하세요:  7
odd
```

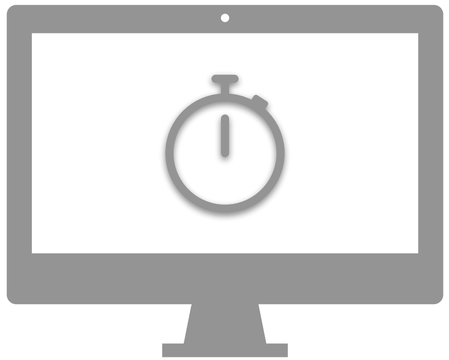
Lab : 팔굽혀펴기 시험

- 팔굽혀펴기 시험 결과 출력하기
  - 중학교에서 팔굽혀펴기 시험을 보는데 채점 기준은 다음과 같다
    - 남학생은 10번 이상을 해야 '합격'이고 아니면 '불합격'
    - 여학생은 8번 이상을 해야 '합격'이고 아니면 '불합격'

  - 실행 결과 예시

```
성별(F/M)을 입력하세요: f
팔굽혀펴기 횟수를 입력하세요: 9
합격

성별(F/M)을 입력하세요: M
팔굽혀펴기 횟수를 입력하세요: 9
불합격

성별(F/M)을 입력하세요: k
팔굽혀펴기 횟수를 입력하세요: 7
성별(F/M) 입력이 잘못되었습니다.
```

간편 조건문



Lab : 조건문을 간편 조건문으로 변환

- 조건문을 간편 조건문으로 변환하기
  - 다음 코드를 간편 조건문으로 바꿔보자

```
margin = False

if margin:
    width = 100 + 10
else:
    width = 100 + 0

print(width)
```

  - 실행 결과 예시

```
100
```

Lab : 간편 조건문

- 간편 조건문 만들기
  - 두 수를 입력받아
    - 두 수가 같은 값이면 '둘은 같은 숫자입니다.'를,
    - 다른 값이면 '둘은 다른 숫자입니다.'를 출력하는 조건문을
  - 간편 조건문으로 작성

- 실행 결과 예시

```
숫자를 입력하세요: 5
숫자를 입력하세요: 9
둘은 다른 숫자입니다.
```

Lab : 간편 중첩 조건문

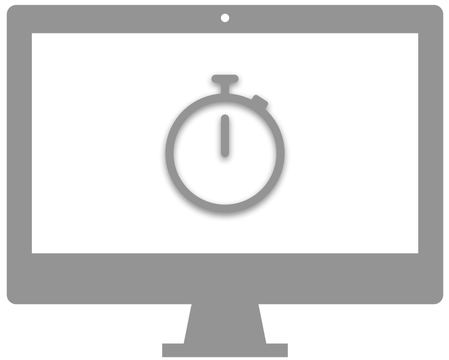
- 중첩 조건문을 간편 조건문(조건 표현식)으로 변환하기
  - 다음 코드를 간편 조건문으로 바꿔보자

```
prompt = '정수를 입력하세요: '
result = '둘 중 더 큰 수는 {}입니다.'
x = int(input(prompt))
y = int(input(prompt))

if x == y:
    print('같은 숫자군요.')
else:
    if x > y:
        print(result.format(x))
    else:
        print(result.format(y))
```

  - 실행 결과 예시

```
정수를 입력하세요:  5
정수를 입력하세요:  -7
둘 중 더 큰 수는 5입니다.
```

while 문



Lab : while-else 문

- while-else 문 작성하기
  - 변수 number를 0으로 초기화
  - number가 10보다 작으면 '10보다 작습니다.'를 출력하면서, number를 1씩 증가
  - number가 10 이상이 되면 '종료'를 출력하면서 while문을 빠져나감

  - 실행 결과 예시

```
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
10보다 작습니다.
종료
```

Lab : 홀수만 출력하는 while-else 문

- 홀수만 출력하는 while-else 문 작성하기
  - 정수 1부터 10까지 포함하고 있는 튜플이나 리스트를 생성
  - while 문으로 앞서 만든 튜플이나 리스트를 순환해서 홀수만 출력
  - 튜플이나 리스트의 객체를 모두 순회한 후 '종료'를 출력하면서 while 문을 빠져나감

- 실행 결과 예시

```
1
3
5
7
9
종료
```

Lab : 특정 문자열을 출력하는 while-else 문

- 조건에 맞는 문자열만 출력하는 while-else 문 작성하기
  - 아래 문자열을 담고 있는 튜플이나 리스트를 생성
    - 'red', 'blue', 'pink', 'green', 'yellow', 'purple'
  - 튜플이나 리스트가 담고 있는 문자 중 'p'로 시작하는 문자열만 출력
  - 튜플이나 리스트의 객체를 모두 순회한 후 "더 이상 'p'로 시작하는 문자열이 없습니다."를 출력하면서 while 문 종료

- 실행 결과 예시

```
pink
purple
더 이상 'p'로 시작하는 문자열이 없습니다.
```

Lab : 최소공배수 구하는 while-else 문

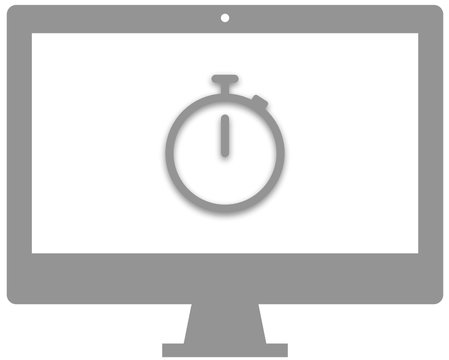
- while-else 문으로 최소공배수 구하기
  - 서로 다른 임의의 정수 3개를 입력 받는다
  - 이 세 정수의 최소공배수를 찾아내고, '최소공배수는 X입니다.'를 출력하면서 종료
    - X는 여기서 찾아낸 최소공배수를 의미
  - 최소공배수(least common multiple)란 2개 이상의 수의 공배수 가운데서 최소인 수
    - 공배수란 두 개 이상 자연수의 공통인 배수
    - 예) 2, 3, 9의 최소공배수는 18

센티널 변수 i에 입력받은 세 수로 각각 나누어 나머지가 모두 0이 될 때까지 연산

  - 실행 결과 예시

```
첫번째 정수를 입력하세요: 2
두번째 정수를 입력하세요: 3
세번째 정수를 입력하세요: 9
최소공배수는 18입니다.
```

for 문 기본 형식



Lab : for 문과 순회형

- for 문에서 다양한 순회형 사용하기
  - 다음과 같은 6가지 자료형을 생성
    - `t = 0, 1, 2, 3, 4`
    - `L = [5, 6, 7, 8, 9]`
    - `s = set('가나다라마')`
    - `d = {'a': 1,'b': 2,'c': 3, 'd': 4, 'e': 5}`
    - `text = 'Hello Python!'`
    - `num = 10`
  - 각 자료형을 for 문에 적용하여, for 문의 전달인자를 출력
    - for 문의 전달인자는 for 변수 in 순회형:에서 변수를 의미
    - 출력 형식은 자유롭게 선택
  - 자료형에 따라 어떠한 전달인자가 선택되는지 비교

Lab : 원하는 만큼 반복하는 for 문

- 원하는 만큼 반복하는 for 문 작성하기
  - 임의의 정수 *n*을 입력받음
  - print 함수로 높이와 밑변을 *n*으로 갖는 직각 삼각형 그리기

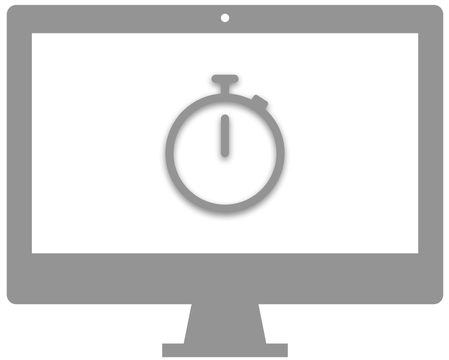
range 클래스를 활용

- print('#' * 정수) 형식으로 출력

- 실행 결과 예시

```
> python right_triangle.py
반복 횟수를 입력하세요: 5
#
# #
# # #
# # # #
# # # # #
```

for 문 일반 형식



Lab : for-else 문과 range 클래스

- range 클래스를 사용하여 for-else 문 작성하기
  - for-else 문에서 range 클래스를 사용해서 0에서 9까지 정수를 하나씩 가져와 10을 곱한 결괏값을 출력
  - 생성한 값을 모두 순회한 후 '종료'를 출력하면서 for 문을 종료
  - 실행 결과 예시

```
0
10
20
30
40
50
60
70
80
90
종료
```

Lab : 홀수만 출력하는 for-else 문

- 홀수만 출력하는 for-else 문 작성하기
  - while-else 문 대신 for-else 문을 사용해서 홀수만 출력하는 순환문 구현
  - 정수 1부터 10까지 포함하고 있는 튜플이나 리스트를 생성
  - for 문으로 튜플이나 리스트를 순환하면서 홀수만 출력

    홀수는 i % 2로 구할 수 있다
  - 튜플이나 리스트의 객체를 모두 순회한 후 '종료'를 출력하면서 for 문을 빠져나간다

  - 실행 결과 예시

```
1
3
5
7
9
종료
```

Lab : 특정 문자열을 출력하는 for-else 문

- 조건에 맞는 문자열만 출력하는 for-else 문 작성하기
  - while-else 문 대신 for-else 문을 사용해서 조건에 맞는 특정 문자열 출력하는 순환문 구현
  - 아래 문자열을 담고 있는 튜플이나 리스트를 생성
    - 'red', 'blue', 'pink', 'green', 'yellow', 'purple'
  - 'p'로 시작하는 문자열만 출력
  - 튜플(또는 리스트)의 객체를 모두 순회한 후 "더 이상 'p'로 시작하는 문자열이 없습니다."를 출력하면서 for 문 종료
  - 실행 결과 예시

```
pink
purple
더 이상 'p'로 시작하는 문자열이 없습니다.
```

Lab : 중첩 for 문

- 중첩 for 문 작성하기
  - 다음과 같이 자료형을 구성
    - `color = ['하얗고', '까맣고', '빨갛고', '노랗고']`
    - `taste = ['달콤한', '짭짤한', '매콤한', '고소한']`
    - `food = ['라면', '피자', '치킨', '햄버거']`
  - `color` + `taste` + `food` 형식으로 표현 가능한 모든 음식을 출력
    - 예) 빨갛고 달콤한 햄버거, 까맣고 고소한 치킨 … 등
  - 실행 결과 예시

```
하얗고 달콤한 라면
하얗고 달콤한 피자
하얗고 달콤한 치킨
하얗고 달콤한 햄버거
하얗고 짭짤한 라면
하얗고 짭짤한 피자
...
노랗고 매콤한 치킨
노랗고 매콤한 햄버거
노랗고 고소한 라면
노랗고 고소한 피자
노랗고 고소한 치킨
노랗고 고소한 햄버거
```

Lab : 2차원 리스트

- 2차원 리스트를 생성하고 출력하는 중첩 for 문 작성하기
  - 알파벳 대문자(A~O)로 이루어진 3행(row) 5열(column)의 2차원 리스트를 생성한 후 출력
    - 각 행(리스트)이 5개의 데이터를 가지고 있는 중첩 리스트
  - 2차원 리스트 각 행의 대문자를 모두 소문자로 바꾸어 출력
    - 행 별로 한 줄씩 소문자로 출력
  - 실행 결과 예시

```
[['A', 'B', 'C', 'D', 'E'], ['F', 'G', 'H', 'I', 'J'], ['K', 'L', 'M', 'N', 'O']]

a b c d e
f g h i j
k l m n o
```

Lab : 구구단 출력

- 구구단을 출력하는 `for` 문 작성하기
  - `for` 문과 `range` 클래스를 활용한 구구단 출력
  - 아래 실행 결과 예시와 같은 형식으로 구구단을 출력

  - 실행 결과 예시

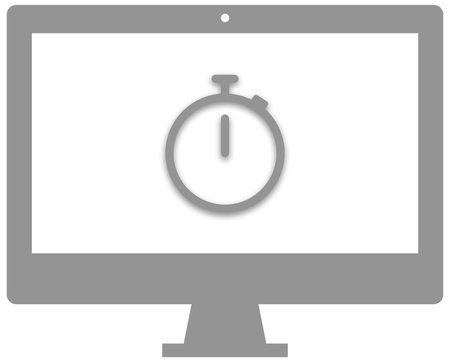
```
2 × 1 = 2
2 × 2 = 4
...
2 × 9 = 18

3 × 1 = 3
3 × 2 = 6
...
3 × 9 = 27

...

9 × 1 = 9
9 × 2 = 18
...
9 × 9 = 81
```

Lab : 윤년 계산기

- for 문으로 윤년 계산기를 구현하기
  - for 문에서 range 클래스를 활용해 윤년 계산하기
  - 윤년 계산 규칙은 다음과 같음(https://ko.wikipedia.org/wiki/윤년)
    - 그 해의 숫자가 4로 나누어 떨어지는 해는 윤년으로 한다
      - 예) 1988년, 1992년, 1996년, 2004년, 2008년, 2012년 …
    - 그 해의 숫자가 4로 나누어지지만 100으로도 나누어지면 평년으로 한다
      - 예) 1900년, 2100년, 2200년, 2300년, 2500년 …
    - 그 해의 숫자가 100으로 나누어지지만 400으로도 나누어지면 윤년으로 한다
      - 예) 1600년, 2000년, 2400년 …
  - 실행 결과 예시
    - 2000년에서 3000년 사이(둘 다 포함)

```
[2000, 2004, 2008, 2012, 2016, 2020, 2024, 2028, 2032, 2036, 2040, 2044, 2048, 2052, 2
056, 2060, 2064, 2068, 2072, 2076, 2080, 2084, 2088, 2092, 2096, 2104, 2108, 2112, 211
6, 2120, 2124, 2128, 2132, 2136, 2140, 2144, 2148, 2152, 2156, 2160, 2164, 2168, 2172,
2176, 2180, 2184, 2188, 2192, 2196, 2204, 2208, 2212, 2216, 2220, 2224, 2228, 2232, 22
36, 2240, 2244, 2248, 2252, 2256, 2260, 2264, 2268, 2272, 2276, 2280, 2284, 2288, 2292
, 2296, 2304, 2308, 2312, 2316, 2320, 2324, 2328, 2332, 2336, 2340, 2344, 2348, 2352,
2356, 2360, 2364, 2368, 2372, 2376, 2380, 2384, 2388, 2392, 2396, 2400, 2404, 2408, 24
12, 2416, 2420, 2424, 2428, 2432, 2436, 2440, 2444, 2448, 2452, 2456, 2460, 2464, 2468
, 2472, 2476, 2480, 2484, 2488, 2492, 2496, 2504, 2508, 2512, 2516, 2520, 2524, 2528,
2532, 2536, 2540, 2544, 2548, 2552, 2556, 2560, 2564, 2568, 2572, 2576, 2580, 2584, 25
88, 2592, 2596, 2604, 2608, 2612, 2616, 2620, 2624, 2628, 2632, 2636, 2640, 2644, 2648
, 2652, 2656, 2660, 2664, 2668, 2672, 2676, 2680, 2684, 2688, 2692, 2696, 2704, 2708,
2712, 2716, 2720, 2724, 2728, 2732, 2736, 2740, 2744, 2748, 2752, 2756, 2760, 2764, 27
68, 2772, 2776, 2780, 2784, 2788, 2792, 2796, 2800, 2804, 2808, 2812, 2816, 2820, 2824
, 2828, 2832, 2836, 2840, 2844, 2848, 2852, 2856, 2860, 2864, 2868, 2872, 2876, 2880,
2884, 2888, 2892, 2896, 2904, 2908, 2912, 2916, 2920, 2924, 2928, 2932, 2936, 2940, 29
44, 2948, 2952, 2956, 2960, 2964, 2968, 2972, 2976, 2980, 2984, 2988, 2992, 2996]
```

for 문과 딕셔너리

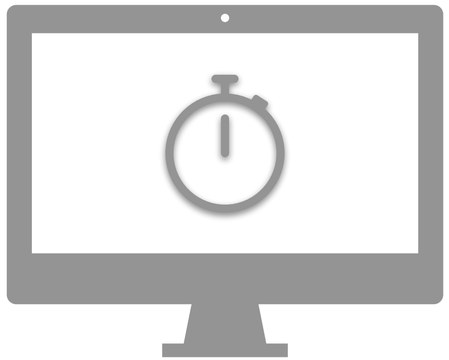


Lab : 딕셔너리 항목 필터링

- 딕셔너리 항목을 필터링하는 for 문 작성하기
  - 아래 직원들 중 연봉이 50,000이상인 직원들의 이름을 딕셔너리를 활용해 출력
  - 직원 연봉 정보
    - David : 30,000
    - James : 50,000
    - Andrew : 45,000
    - Rita : 70,000
    - Michael : 10,000
  - 실행 결과 예시

```
James's salary is 50,000
Rita's salary is 70,000
```

무한 루프와 break/continue



Lab : for 문 흐름 제어

- for 문 흐름 제어하기
  - 임의의 정수로 구성한 튜플이나 리스트를 만들어 변수에 할당한 후 그 변수를 출력
    - 예) `1, 4, 7, 3, 4, 6, 0, 1, 4, 11, 10, 5, ...`
  - for 문을 활용하여 변수가 담고 있는 값들을 순서대로 출력
    - 이 때, 숫자가 `5` 이상이면 출력하고, 그렇지 않으면 출력하지 않는다
    - 그런데 숫자가 `0`이면, 즉시 for 문을 종료
  - 반드시 `break`와 `continue` 명령어 둘 다를 사용해야 한다
  - 실행 결과 예시

```
[1, 4, 7, 3, 4, 6, 0, 1, 4, 11, 10, 5]

7
6
0 : for문을 빠져나갑니다.
```