3. 빅데이터 처리를 위한 파이썬 기초 02

흐름 제어(Flow Control)

- 조건 및 반복 처리 -

어떤 반복의 방법을 선택할까?

- for 문
  - 횟수 제어
  - 주어진 횟수만큼 반복
  - 시퀀스는 리스트나 문자열과 같은 열거 데이터 타입
  - 시퀀스에 있는 항목을 하나씩 가져와서 반복을 한다.
  - 시퀀스에 더 이상 가져올 항목이 없으면 반복을 종료.

for 변수 in 시퀀스 :
    반복 실행할 문장

- while 문
  - 조건 제어
  - 특정한 조건이 만족되면 계속 반복

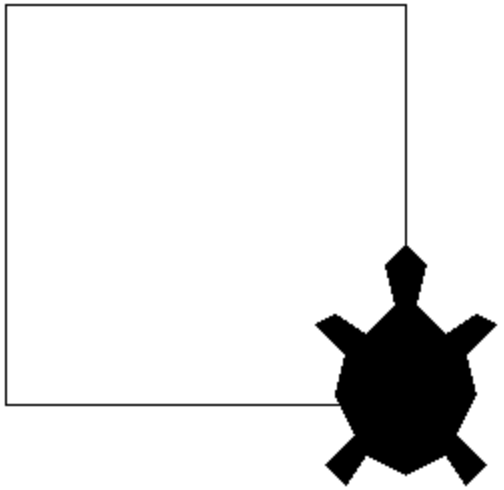
while 조건식 :
    조건이 참인 경우 실행할 문장

정사각형 그리기

- i는 0부터 3까지 값을 가지게 됨
- range( ) 는 주어진 범위의 수를 생성하는 함수

```
>>> import turtle as t
>>> t.shape("turtle")
>>> t.shapesize(3)
>>> t.pensize(5)
>>> t.pencolor("red")
>>> t.forward(200)
>>> t.left(90)
>>> t.forward(200)
>>> t.left(90)
>>> t.forward(200)
>>> t.left(90)
>>> t.forward(200)
>>> t.left(90)
>>> t.clear()
```

```
>>> for i in range(4) :
        t.forward(200)
        t.left(90)
```



정다각형 그리기

- 사용자가 요구하는 정다각형 그리기

```
import turtle as t
t.color("purple")

t.begin_fill()

for x in range(5):
    t.forward(50)
    t.left(360/5)

t.end_fill()
```

몇 각형을 그릴까요? 5

정다각형 그리기

- 사용자가 요구하는 정다각형 그리기

```
import turtle as t
t.color("purple")

n = int(input("몇 각형을 그릴까요? "))

t.begin_fill()

for x in range(n):
    t.forward(50)
    t.left(360/n)

t.end_fill()
```

한 선의 길이는 얼마가 좋을까요?
100

반복 제어에 대한 이해 : 구구단 출력

- 2단 출력

```
for i in range(1, 10, 1) :
    print("2 * ", i, " = " , 2 * i)
```

```
2 * 1  =  2
2 * 2  =  4
2 * 3  =  6
2 * 4  =  8
2 * 5  =  10
2 * 6  =  12
2 * 7  =  14
2 * 8  =  16
2 * 9  =  18
```

```
print("2 * " + str(i) +  " = " +  str( 2 * i))
```

반복 제어에 대한 이해 : 구구단 출력

- 2단부터 9단까지 출력(중첩 for)

```
for i in range(1, 10, 1) :
    print("2 *", i, " = " , 2 * i)
print("₩n")

for i in range(1, 10, 1) :
    print("3 *", i, " = " , 3 * i)
print("₩n")

for i in range(1, 10, 1) :
    print("4 *", i, " = " , 4 * i)
print("₩n")
.
.
.
```

```
for dan in range(2, 10) :
    for i in range(1, 10) :
        print(dan, " * ", i, " = ", dan*i)

    print("₩n")
```

반복 제어에 대한 이해 : 리스트

- 리스트 안에 저장된 데이터 하나씩 출력.

```
student_info = [ '컴과', 4, 2022011023, '김은경', 156.8 ]

for i in range( len(student_info) ) :
    print(i)
```

조건 분기문

- if 문

```
if   조건식   :
     조건이 참인 경우 실행할 문장
```

- if ~ else문

```
if   조건식   :
     조건이 참인 경우 실행할 문장
else :
     조건이 참이 아닌 경우 실행할 문장
```

조건의 영향을 받는 문장들은 반드시 들여쓰기를 해야한다.
들여쓰기는 통상적으로 4개의 스페이스로 처리한다.

조건 분기문

- if ~ elif ~ else문

```
if   조건식1  :
     조건1이 참인 경우 실행할 문장
elif   조건식2  :
     조건2이 참인 경우 실행할 문장


else :
     조건이 참이 아닌 경우 실행할 문장
```

조건에 대한 이해 : 합격/불합격 판단

- 성적이 60점 이상이면 합격으로,
  60점 미만이면, 불합격으로 처리해야한다면

[코드]

```
score = int(input("성적을 입력하시오: "))

if score >= 60 :
    print("합격입니다. ")
else :
    print("불합격입니다.")
```

파이썬 인터프리터에게
아직 전체 문장이 끝나지 않았으니
잠시 해석을 미뤄달라는 의미의 기호

```
성적을 입력하시오: 80
합격입니다.
```

조건에 대한 이해 : 홀수/짝수 판단

```
num = int(input("정수를 입력하시오: "))

if num % 2 == 0 :
        print("짝수입니다.")
else:
        print("홀수입니다.")
```

```
정수를 입력하시오: 10
짝수입니다.
```

조건에 대한 이해 : 양수, 0, 음수 판단하기

- 입력 받은 수가 양수이면 “양수”, 0이면 “0”, 음수이면 “음수”라고 출력해 주는 코드를 작성해 봅시다.

```
while True :
    n = input("정수를 입력하세요")

    if n == "Q" or n =="q" :
        break

    n = int(n)

    if n >0 :
            print("양수")
    elif n == 0 :
            print("0")
    else :
            print("음수")
```

양수, 음수, **0** 판단 부분

in 을 이용한 조건 검사

- in 을 사용하여 문자열이나 리스트의 특정 요소가 있는지 검사할 수 있다.

```
if  'o'  in  "I love gnu" :
    print("'I love gnu'에 'o'가 포함되어 있다.")

s  =  "I love gnu"
if  'v'  in  s :
    print("'I love gnu'에 'v'가 포함되어 있다.")

s_list = [10, 20, 30, 40]
if 5 in s_list :
    print("리스트에 5가 포함되어 있다.")
```

in 을 이용한 반복 처리

- in 을 사용하여 문자열이나 리스트의 특정 요소를 처리할 수 있다.

```
for i in "아름다운 우리대학" :
    print(i)
```

```
아
름
다
운

우
리
대
학
```

in 을 이용한 반복 처리

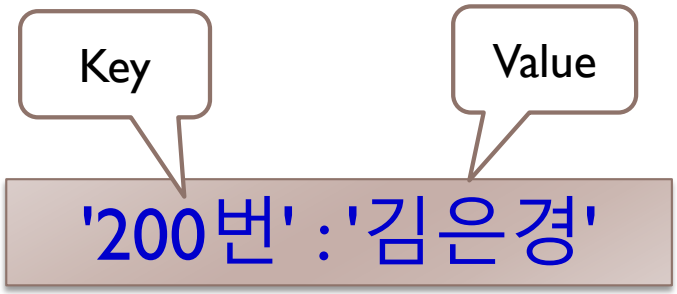
- in 을 사용하여 문자열이나 리스트의 특정 요소를 처리할 수 있다.

```
hap =0
i =1

for  e  in  [10, 20, 30, 40, 50] :
    print("%d 번째 수는 %d"  %(i, e))
    i += 1
    hap += e

print("총 합은 %d 이다."  %(hap))
```

```
1 번째 수는 10
2 번째 수는 20
3 번째 수는 30
4 번째 수는 40
5 번째 수는 50
총 합은 150 이다.
```

in 으로 딕셔너리의 요소에 반복 접근

```
dic = {'100번':'박순화', '200번':'김은경'}
for k in dic :
    print(k, dic[k])
```




```
100번 박순화
200번 김은경
```

* 사전의 각 요소의 키 값을 **k**로 가져와서 처리함

남자/여자 판단 간단한 문제 해결부터

- 주민등록번호를 입력 받아서 남자인지 여자인지 판단하는 프로그램을 작성해 보자.

```
num = input("주민등록번호를 입력하세요 : ")

if num[7] == "1" :
    print("당신은 남자입니다.")
else :
    print("당신은 여자입니다.")
```

```
주민등록 번호를 입력하시오: 890105-8929387
당신은 여자입니다.
```

남자/여자 판단

조금 넓은 범위의 문제 해결

```
num = input("주민등록번호를 입력하세요 : ")

if num[7] == "1" or num[7] == "3":
    print("당신은 남자입니다.")
elif num[7] == "2" or num[7] == "4":
    print("당신은 여자입니다.")
else :
    print("판단할 수 없습니다. 다시 입력하세요")
```

주민등록 번호를 입력하시오: 890105-8929387
판단할 수 없습니다. 다시 입력하세요.

남자/여자 판단 좀더 완전한 문제해결

주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] :
길이가 맞지 않습니다. 다시 입력하세요
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 90
길이가 맞지 않습니다. 다시 입력하세요
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 901205-
길이가 맞지 않습니다. 다시 입력하세요
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 940102-2315698
당신은 여자입니다.
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 780102-1398321
당신은 남자입니다.
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 890926-3795684
당신은 남자입니다.
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : 980126-5698456
판단할 수 없습니다. 다시 입력하세요.
주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : Q
프로그램 종료

다음과 같이 동작 하도록 처리해 보자

남자/여자 판단 좀더 완전한 문제해결 : 소스코드

```
while  True :
    num = input("주민등록번호를 입력하세요[종료는 'Q' 또는 'q'] : ")

    if num == "Q" or num == "q" :
        print("프로그램 종료")
        break

    if len(num) != 14 :
        print("길이가 맞지 않습니다. 다시 입력하세요")

    elif num[7] == "1" or num[7] == "3":
        print("당신은 남자입니다.")

    elif num[7] == "2" or num[7] == "4":
        print("당신은 여자입니다.")

    else :
        print("판단할 수 없습니다. 다시 입력하세요.")
```

로그인 프로그램 간단한 문제 해결부터

- 아이디와 패스워드를 입력 받고 본인 확인을 해주는 프로그램을 만들어 보자. 일단 직관적으로 작성해 보자.
  - 허용된 아이디 : jslee
  - jslee의 암호 : 1234

```
name = input("아이디를 입력하세요 : ")
passwd = input("비번을 입력하세요 : ")

if( name == "jslee"  and  passwd =="1234") :
        print("로그인 성공")
else :
        print("로그인 실패")
```

로그인 프로그램 좀 더 실용적인 해결

- 아이디와 패스워드가 일치할 때까지 반복적으로 입력 받도록 업그레이드 해 봅시다.

 # 무한 루프가 해결책이 될까?

 # 종료 조건은 어떻게 할까?

```
while(True) :
    name = input("아이디를 입력하세요 : ")
    passwd = input("비번을 입력하세요 : ")
    if( name == "jslee"  and  passwd =="1234") :
            print("로그인 성공 ₩n")
    else :
            print("로그인 실패 ₩n")
```

로그인 프로그램

좀 더 실용적인 해결 방법

- 이름과 암호 모두 성공적으로 입력할 때까지 반복적으로 입력 받도록 하면

```
name=""
passwd=""

while(name != "jslee"   or   passwd !="1234") :
    name = input("아이디를 입력하세요 : ")
    passwd = input("비번을 입력하세요 : ")

print("₩n로그인 성공")
```

로그인 프로그램

좀 더 친절한 프로그램

로그인 정보를 입력하세요
아이디를 입력하세요 : gggg
비번을 입력하세요 : 12

로그인 실패입니다. 다시 정보를 입력하세요
아이디를 입력하세요 : jsee
비번을 입력하세요 : 12

로그인 실패입니다. 다시 정보를 입력하세요
아이디를 입력하세요 : jslee
비번을 입력하세요 : 1234

로그인 성공
>>>

로그인 프로그램 좀 더 친절한 프로그램

```
name=""
passwd=""
flag=0    #로그인 정보가 제대로 입력되지 않은 상태

while(name != "jslee"  or  passwd !="1234") :
    if flag == 0 :
        print("₩n로그인 정보를 입력하세요")
        flag =1
    else :
        print("₩n로그인 실패입니다. 다시 정보를 입력하세요")

    name = input("아이디를 입력하세요 : ")
    passwd = input("비번을 입력하세요 : ")

print("₩n로그인 성공")
```

예외 처리

(ecxeption handling)

예외 처리

- **예외(exception)**
  - 프로그램 실행 중에 발생할 수 있는 비정상적인 사건
  - 예) 오버플로, 언더플로, 0으로 나누기, 배열 첨자 범위 이탈 오류, EOF(end-of-file) 조건 등

- **예외 처리(exception handling)**
  - 예외가 탐지되었을 때 프로그램의 중단 없이 적절한 행동을 취해서 다시 정상적으로 실행되도록 하는 메커니즘
  - 예외를 처리하는 부분을 예외 처리기(exception handler)라 함

예외 처리

- **예외의 원인은 무엇인가?**
  - H/W 문제
  - OS의 설정의 실수
  - 라이브러리의 손상
  - 사용자의 입력 실수
  - 받아들일 수 없는 연산
  - 할당되지 않은 기억장치 접근 등

예외 처리

- **예외 처리 기능이 있는 언어**
  - 1960년대 PL/I 에 처음 도입, 1970년대 CLU에 의해 크게 발전
  - Ada, C++, Java, Common LISP 등이 예외처리 제공

- **예외 처리 기능이 없는 언어 : Pascal, C, Fortran**
  - 프로그래머가 예외에 대한 검사를 사전에 수행해야 함

```
if (num == 0)
    printf("error : divide by zero₩n");
else
    average= sum / num;
```

예외에 대한 조건 처리와 정상적인 조건 처리를 구분하기가 어렵다는 문제점 있음

파이썬의 예외 처리 : try ~ except

```
a = "hello"
print(a)
print(a[10])
print(a)
```

```
hello
Traceback (most recent call last):
  File "D:/exception.py", line 4, in <module>
    print(a[10])
IndexError: string index out of range
>>>
```

예외 처리가 없으면 에러 발생 후 프로그램 종료

```
try :
    a = "hello"
    print(a)
    print(a[10])
    print(a)

except :
    print("error 발생")
```

```
hello
error 발생
```

예외 발생 후 프로그램 실행이 계속 될 수 있음

파이썬의 예외 처리 : try ~ except

try :
    예외 발생 가능 코드

except 예외타입 :
    예외 발생시 실행되는 코드

에러 타입에 따른 예외 처리

```
try :
    a = 10
    b = 0
    print(a)
    print(b)
    print(a/b)

except :
    print("error 발생")
```

왼쪽의 코드에 예외 처리가 없다면 발생할 오류

```
Traceback (most recent call last):
  File "D:/exception.py", line 6, in <module>
    print(a/b)
ZeroDivisionError: division by zero
```

```
10
0
error 발생
```

예외 처리는 해서 비정상적 종료는 하지 않았지만, 무엇이 문제인지 알 수 없음

따라서 에러 타입에 따른
예외 처리를 할 필요가 있다!!!

에러 타입에 따른 예외 처리

```
try :
    a = 10
    b = 0
    c= "jslee“

    print(a)
    print(b)
#   print(a/b)
#   print(c[10])

except  ZeroDivisionError :
    print("ZeroDivisioError error 발생")

except  IndexError :
    print("IndexError error 발생")
```

* 왼쪽의 코드에서 주석 부분을 한 줄씩 살려서 실행해 보라. 결과는?

파이썬의 예외 처리 : try ~ except

try :
    예외 발생 가능 코드

except 예외타입 **:**
    예외 발생시 실행되는 코드

else :
    예외 발생하지 않았을 때 실행되는 코드

finally :
    예외 발생 여부와 상관없이 실행되는 코드

에러 타입에 따른 예외 처리

```
try:
    a = 10
    b = 0
    print(a)
    print(b)
    print(a/0)

except IndexError :
    print('Index error 발생')

except ZeroDivisionError :
    print('ZeroDivision error 발생')

else :
    print('try 영역이 정상적으로 실행됐을 때')

finally :
    print('최종적으로 무조건 실행되는 구문')
```

- try에서 에러가 발생하면
  - except 가 실행 -> finally 실행
- 에러가 발생하지 않으면
  - try -> else -> finally 순으로 실행

양수, 0, 음수 판단하기에서 예외처리

- 입력 받은 수가 양수이면 "양수", 0이면 "0", 음수이면 "음수"라고 출력해 주는 코드를 작성해 봅시다.

```
while True :
    n = input("정수를 입력하세요")

    if n == "Q" or n =="q" :
        break

    n = int(n)

    if n >0 :
            print("양수")
    elif n == 0 :
            print("0")
    else :
            print("음수")
```

```
while True :
    n = input("정수를 입력하세요")

    if n == "Q" or n =="q" :
        break

    try :
        n = int(n)

    except ValueError :
        print("정수로 변환할 수 없습니다.")

    else :
        if n >0 :
            print("양수")
        elif n == 0 :
            print("0")
        else :
            print("음수")
```

예외처리 사용 예

- 두 수를 입력 받아서 큰 수를 작은 수로 나눈 결과를 출력하는 프로그램을 작성해 봅시다.
  - if ~ else구문을 이용한 예외 처리
  - try ~ except 를 이용한 예외 처리

참고 : 관계 연산자

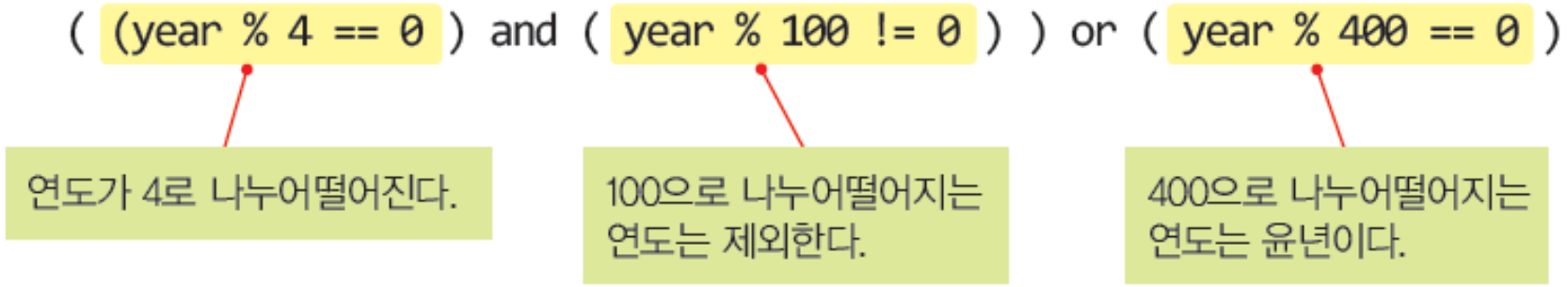
연 산	의 미
x == y	x와 y가 같은가?
x != y	x와 y가 다른가?
x > y	x가 y보다 큰가?
x < y	x가 y보다 작은가?
x >= y	x가 y보다 크거나 같은가?
x <= y	x가 y보다 작거나 같은가?

참고 : 논리 연산자

- 논리 연산자 and, or, not는
  프로그램의 로직 설계에 매우 중요

- and 연산은 양쪽 관계가 모두 참일 때만 True 반환
- or 연산은 양쪽 관계 중 하나라도 True이면 True 반환
- not 연산은 True라면 False로, False라면 True로 반환

논리 연산자의 사용 : 윤년 판단하기



year의 값이 4의 배수이면서 100의 배수는 아니거나
또는
400의 배수이면 윤년이다.