파이썬 기초

❖ 변수명 만들기

```
_myname = 'hong'
my_name = '홍길동'
MyName2 = 'Hong gil-dong'
counter = 1
Counter = 2
```

- 규칙
  - 첫 문자는 밑줄 또는 영문자로 시작
  - 잘못된 예
    - 1_unit, %var, @address
  - 대소문자 구분
  - 예약어는 사용할 수 없음

```
# 예약어 확인
import keyword
keyword.kwlist
```

변수명 만들기

❖ 변수명 지정시 주의사항

- 기존 함수명에 값을 대입하는 경우
  - 함수에서 변수로 변경됨(에러가 아님)

```
>>> abs(-1)
1
>>> abs = 1
>>> abs(-3)
에러 발생
```

변수에 값 대입하기

❖ 변수 사용

```
number1 = 1                 // 정수
pi = 3.14                   // 실수
flag = True                 // 부울린
char = 'x'                  // 문자
chars = 'I love Python'     // 문자열
```

- 변수를 선언없이 사용 가능
- 데이터 타입 고정되어 있지 않음
  - 대입할 때 값의 타입에 따라 그때 그때 결정

주석 처리하기(#)

❖ 주석

```
# 주석 처리 예시임
# 작성일 : 2017.08.18
a = 1         # a에 1을 대입함
b = 5         # b에 5를 대입함
print(a+b)    # a+b의 결과를 출력함
```

- #
  - 이후 문자들은 주석으로 처리됨
  - 한 줄을 주석으로 만듦
- """ 블록 """ (또는 ''' 블록 ''')
  - 삼중 따움표로 여러줄에 걸친 블록을 주석 처리할 수 있음

```
# 삼중 따움표로 특정 영역 주석 처리
"""a = 1
b = 5
print(a+b)"""
a = 2
b = 6
print(a+b)
```

❖ 파이썬 자료형

```
int_data = 1                            # 정수 선언
float_data = 3.14                       # 실수 선언
complex_data = 1+5j                     # 복소수 선언
str_data1 = 'I love Python'             # 문자열 선언
str_data2 = '안녕하세요'                  # 문자열 선언
list_data = [1, 2, 3]                   # 리스트 선언
tuple_data = (1, 2, 3)                  # 튜플 선언
dict_data = {0:'False', 1:'True'}   # 사전 선언
```

- 수치형
  - 정수
  - 실수
  - 복소수
- 문자열
- 리스트
- 튜플
- 사전

자료형 출력 (print)

❖ print( ) 함수

```
a = 200
msg = 'I love Python'
list_data = ['a', 'b' 'c']
dict_data = { 'a':97, 'b':98}
print(a)
print(msg)
print(list_data)
print(list_data[0])
print(dict_data)
print(dict_data['a'])
```

○ 실행 결과

```
200
I love Python
['a', 'bc']
a
{'a': 97, 'b': 98}
97
```

❖ print( ) 함수

- 입력된 값을 화면에 출력한 후 자동 줄바꿈
  - 마지막에 '\n' 추가
- 줄바꿈 하지 않는 경우 end=''를 매개변수로 전달
  - 마지막 출력 문자를 빈문자('')로 지정

```
print('#', end='')
print('#')
```

❖ 들여쓰기

- 실행 코드 부분을 지정하는 { } 괄호 없음
  - 다른 언어의 if, while, for 문 블록지정 괄호
- 규칙
  - 가장 바깥쪽의 실행 코드는 들여쓰기 없이 시작해야 함
    - 들여쓰는 경우 에러
  - 콜론(':') 다음 줄부터 시작하는 실행 코드는 들여쓰기 간격이 모두 동일해야 함

```
list_data = [1, 2, 3]
if 1 in list_data:
   print('1이 list_data에 있습니다')
      print(list_data)       # 오류발생
else:
   print('a가 list_data에 없습니다')
```

❖ if~else

```
if 조건:
    조건이 참(True)일 경우 실행할 코드
else:
    조건이 거짓(False)일 경우 실행할 코드

# else 생략 가능
if 조건:
    조건이 참(True)일 경우 실행할 코드
```

```
x = 1
y = 2
if x>=y:
    print('x가 y보다 크거나 같습니다.')
else:
    print('x가 y보다 작습니다')
```

❖ if ~ elif ~ else

```
if 조건1:
    조건1이 참(True)일 경우 실행할 코드
elif 조건2:
    조건2가 참(True)일 경우 실행할 코드
    :
elif 조건n:
    조건n이 참(True)일 경우 실행할 코드
else:
    모든 조건이 거짓(False)일 경우 실행할 코드
```

```
x = 1
y = 2
if x>y:
    print('x가 y보다 큽니다.')
elif x<y
    print('x가 y보다 작습니다.')
else:
    print('x와 y가 같습니다. ')
```

for 문

❖ for 문

```
for 변수 in 범위:
    반복으로 실행할 코드
```

- 범위에 올 수 있는 것
  - 문자열
  - 리스트나 튜플
  - 사전
  - range()
  - 그 외 반복 가능한 객체
- 범위에 있는 값들을 루프를 돌 때마다 변수에 저장

for 문

❖ 문자열을 범위로

```
str = 'abcdef'
for c in str:
    print(c)
```

```
a
b
c
d
e
f
```

for 문

❖ 리스트를 범위로

```
scope = [1, 2, 3, 4, 5]
for x in scope:
    print(x)
```

```
1
2
3
4
5
```

for 문

❖ 사전을 범위로

```
ascii_codes = {
    'a' : 97,
    'b' : 98,
    'c' : 99
}
for code in ascii_codes:
    print(code)     # 사전의 키값 출력
```

```
a
b
c
```

for 문

❖ range() 함수를 범위로

```
for c in range(10):
    print(c)
```

```
0
1
2
3
4
5
6
7
8
9
```

- range(10)
  - 0부터 10미만 까지 (0 ~9)

for ~ continue ~ break

❖ for 반복문 제어

```
for 변수 in 범위:
    …
    continue     # 다음 반복문 수행
    …
    break        # for 반복문 탈출
```

```
scope = [1, 2, 3, 4, 5]
for x in scope:
    print(x)
    if x<3:
        continue
    else:
        break;
```

❖ for ~ else

- for 반복문이 break에의해 중단됨 없이 모두 실행된 경우 else 코드 실행

```
for 변수 in 범위:
    반복 실행 코드
else:
    for 반복문이 모두 실행되었을 때 실행할 코드
```

```
scope = [1, 2, 3]
for x in scope:
    print(x)
    break;
else:
    print('perfect')
```

❖ while 문

```
while 조건:
    반복 실행 코드
    continue          # while 구문 처음으로 이동하여 반복문 계속 실행
    …
    break             # while 구문 탈출
```

```
x = 0
while x<10:
    x = x + 1
    if x<3:
        continue
    print(x)
    if x>7:
        break;
```

❖ while 문

```
x = 1
total = 0
while 1:     # 1은 True로 해석 - 무한 루프
   total = total + x
   if total > 100000:
      print(x)
      print(total)
      break
   x = x+1
```

- True : 0이 아닌 값
- False : 0

None

❖ None 상수

- Types.NoneType의 유일한 값
- 값이 존재하지 않는 변수에 대입하여 이 변수에 아무런 값이 없다는 것을 나타내기 위해 주로 활용

```
val = None
condition = 1
if condition == 1:
   val=[1, 2, 3]
else:
   val='I love Ptython'
```

- 어떤 변수가 None인지 여부 판단

```
if val == None:
   …
else:
   …
```