06.Python 제어문과 함수기초

1. Python 제어문

1. 들여쓰기
   - 파이썬은 들여쓰기를 강제하여 코드의 가독성을 높임
   - 가장 바깥쪽의 코드는 반드시 1열에 시작
   - 블록 내부에 있는 statement들은 동일한 열에 위치
   - 블록의 끝은 들여쓰기가 끝나는 부분으로 간주
   - 들여쓰기를 할 때에는 탭과 공백을 섞어 쓰지 않음

1] 적용 예시

```
if a > 1:
   print 'king'
   print end'
```

- 첫번째 라인은 실행하는 함수문의 헤더 역할 수행
- 밑 라인은 함수문의 몸체 역할 수행
- 몸체: 새로운 블록이 시작되는 것으로 새 블록에서 코딩
- 메인 블록과 다르게 새 블록은 반드시 들여쓰기 해야 함
- : 을 넣어 새블록의 시작을 알림
- : 이 아래의 새 블록은 반드시 들여쓰기 필요
- 일반적 메인블록은 반드시 1열에서 시작
- 헤더의 마지막에는 ;(콜론)으로 마무리

1. Python 제어문

2. if문

```
if 조건식1:
   statements
elif 조건식2:
   statements
elif 조건식3:
   statements
else:
   statements
```

- 조건식이나 else 다음에 콜론(:) 표기 필요
- 들여쓰기(indentation)를 잘 지켜야 함
- statements: 조건이 만족될 때 수행되어야 하는 문장들
- elif, else문은 있어도 되고 없어도 되는 문법들
- elif: elseif의 약자
- elif: 위 조건식 1이 만족 안되었다면 elif의 조건식2 확인 후 수행
- elif 문은 여러 개 삽입 가능
- 조건식 1,2,3이 모두 불만족 시 else의 statement 수행

2] 적용 예시

```
n = -2
if n > 0:
    print ('Positive')
elif n < 0:
    print ('Negative')
else: print ('Zero')
```

1. Python 제어문

3. for문

```
for <타겟> in <컨테이너 객체>:
    statements
else:
    statements
```

컨테이너 객체에서 원소를 꺼내 타겟에 삽입
statement는 타겟의 value를 활용하여 코딩

2] 적용 예시

```
l = ['cat', 'dog', 'bird', 'pig']
for k, animal in enumerate(l):
    print (k, animal)
```

결과

```
---------------------------
0 cat
1 dog
2 bird
3 pig
```

l는 리스트 → 리스트 객체
enumerate(l)의 k 자리에 순차번호 , animal 에 리스트의 첫번째 원소가 들어감
enumerate는 ['cat', 'dog', 'bird', 'pig'] 을 반복하며 순차적으로 순회
앞에 있는 0, 1, 2, 3이 그대로 각각의 문자열에 대한 index 역할 수행

range 내장함수가 반환해주는 것은 list
range(10) = [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]

```
for x in range(10):
  print x,
```

결과

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

```
for x in range(2, 4):
    for y in range(2, 10):
        print x, '*', y, '=', x*y
    print
```

결과

```
---------------------------
```

for 루프를 여러 개 중첩하여 사용 가능
x가 2일 때 y가 2, 3, 4, 5, 6, 7, 8, 9 수행
x가 3일 때 y가 역시 2, 3, 4, 5, 6, 7, 8, 9 수행

1. Python 제어문

4. **While** 문

```
sum = 0
a = 0
while a < 10:
    a = a + 1
    sum = sum + a
print sum
```

결과

55

a가 10이 되어 조건 불만족 시 중단되고 현재 누적 값 출력

2] 적용 예시

```
x = 0
while x < 10:
   print x, x = x + 1
else:
   print 'else block'

print 'done'
```

결과

---

0 1 2 3 4 5 6 7 8 9 else block
done

. 콤마가 있으므로 옆으로 값 프린트
. while문 끝내기 직전에 else문 수행
. else문은 while문에서도 쓰일 수 있고, for문에서도 쓰일 수 있음

2. Python 함수 기초

1. **함수의 개념 및 함수 사용법**

가. **함수의 개념**

- 함수는 반복적인 코드를 없에 주어 코드의 길이를 짧게 만들어 준다.
- 코드의 유지보수를 쉽게 만들어 준다
- 연관된 코드들을 여러 번 수행 가능하도록 함수에 엮음
- 코드의 양이 많을 경우 함수를 사용하여 한번에 호출 가능
- 코드의 길이가 짧아지고 코드의 유지보수가 쉬움
- 함수를 정의하는 방식: def 사용 (define의 약자)
- 함수의 인자도 타입형을 선언해 주지 않음

1] 적용 예시

```
def add(a, b):
    return a + b

print add(3, 4)
```

- a 타입 → int, b의 타입 → int
- 인자를 받을 때 타입이 결정되기 때문에 동적인 특징을 가짐
- 7을 리턴 받아서 리턴받은 7의 값을 프린트

2. Python 함수 기초

2. 레퍼런스와 pass

가. 레퍼런스

- 함수 이름에 저장된 레퍼런스를 다른 변수에 할당하여 그 변수를 이용한 함수 호출 가능

```
f = add            # add 변수 → 함수 객체를 가리키고 있는 레퍼런스
print f(4, 5)      # f하고 add가 동일한 함수를 가리키고 있는 식별자
print f
print f is add     #  true , f와 add가 같은 객체인지, 같은 식별자인지 알아보는 키워드
```

나. pass 적용

- 함수의 몸체에는 최소한 한개 이상의 statement가 존재해야 함
- 아무런 내용이 없는 몸체를 지닌 함수를 만들 때에는 pass 라는 키워드를 몸체에 기술 필요

```
def simple():   # 인자를 받는 것이 하나도 없음
    pass        # 내용을 안 적고 싶어도 반드시 적어야 함, pass 예약어 사용

print simple()  # simple 을 호출해서 받아내는 것이 없다. → none 객체 리턴
```

2. Python 함수 기초

3. 함수에서 다른 함수 호출

함수에서 다른 함수 호출 가능

```
def add(a, b):
    return a + b


def myabs(x):       # myabs는 인자를 x로 받아, x가 0보다 작으면 부호를 바꿔줌
    if x < 0:
        x = -x
    return x


def addabs(a, b):  # add 함수 호출후 myabs를 호출 절대값 부호 변경
    c = add(a, b)
    return myabs(c)


print addabs(-5, -7)
```

```
def minus(a, b):
   return a – b

print minus(a=10, b=53)     # 인자의 이름과 함께 인자 값을 넘겨줄 수 있음
print minus(b=17, a=15)     # 인자의 이름을 적어주면 반드시 a값 먼저 안 적어도 괜찮음
```

2. Python 함수 기초

4. 두개 이상의 인자 값 동시 반환 및 동적인 자료형 결정

1) 두개 이상의 인자 값 동시 반환

```
def calc(x, y):
    return x + y, x - y, x * y, x / y   # 인자는 2개인데, 총 4개(더하기, 빼기, 곱하기, 나누기한 값)을 리턴


print calc(10, 2)    # 여러 개의 값이 콤마의 형태로 묶여있는 자료: tuple(튜플)
```

결과

```
---------------------------
(12, 8, 20, 5)
```

2) 동적인 자료형 결정

```
def add(a, b):
    return a + b
c = add(1, 3.4)                  # 정수 + 실수 = 실수
d = add('dynamic', 'typing')  # 문자열 + 문자열 = 문자열
e = add(['list'], ['and', 'list'])   # 리스트 + 리스트 = 리스트
```

결과

```
---------------------------
4.4
dynamictyping
['list', 'and', 'list']
```

3) 재귀적 (Recursive) 함수 호출

- 재귀 함수: 함수 몸체에서 자기 자신을 호출하는 함수
- 수학에서 점화식과 유사한 코드
- 반드시 종결 조건 및 종결 조건이 만족할 때의 반환 값이 존재

```
def sum(N):                    # 1부터 N까지 더하는 재귀 함수
    if N == 1:                 # 종결 조건
        return 1               # 종결 조건이 만족할 때의 반환 값
    return N + sum(N-1) # 재귀 호출

print sum(10)
```

결과

```
---------------------------
55
```