파이썬 프로그래밍

제어문과 함수 기초

한국기술교육대학교
온라인평생교육원

▣ 파이썬 제어문

1. 들여쓰기와 제어문

•파이썬은 들여쓰기를 강제하여 코드의 가독성을 높인다.
•가장 바깥쪽의 코드는 반드시 1열에 시작한다.
•블록 내부에 있는 statement들은 동일한 열에 위치해야 한다.
•블록의 끝은 들여쓰기가 끝나는 부분으로 간주된다.
- python에는 {, }, begin, end 등의 키워드가 존재하지 않는다.
•들여쓰기를 할 때에는 탭과 공백을 섞어 쓰지 않는다.

- 들여쓰기를 강제하여 코드의 가독성 높임

```
a = 1 # 성공
  a = 1 # 실패
```

```
File "<ipython-input-160-0453a5e2fe16>", line 2
  a = 1 # 실패 ^

IndentationError: unexpected indent
```

- a와 b가 같은 column(컬럼) 에 존재 X → 에러가 발생
- a = 1과 b = 2는 같은 블록이 존재하는 코드
- 같은 블록이 존재하면 반드시 들여쓰기가 일치해야 함

▣ 파이썬 제어문

1. 들여쓰기와 제어문

```
if a > 1:
   print 'big'
    print 'really?'
```

```
File "<ipython-input-161-96672a766929>", line 3
  print 'really?' ^ I

ndentationError: unexpected indent
```

- 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
- ; 뒤쪽에서 엔터키를 누르면 들여쓰기로 들어감
- Tap키를 누르면 동일한 컬럼에서 제어문의 몸체가 시작
- f=10 까지가 if절의 블록 = if문의 몸체는 한줄
- g=10은 바깥쪽에 있는 if절의 블록
- backspace나 delete 키를 눌러 뒤쪽으로 이동
- a=1, b=2, if a<10;, 과 x=10은 같은 레벨 → statement가 총 4개
- 들여쓰기가 들쑥날쑥하면 파이썬에서는 '문법적 에러'로 간주
- 들여쓰기가 강제화(문법 안에 들여쓰기가 포함됨)
- '들여쓰기'는 개발자가 가지고 가야 할 중요한 원칙
- 들여쓰기를 하면 → 깔끔하고 이해하기 쉬운 코딩 가능
- 들여쓰기르 올바르게 하지 않으면? → syntax 에러 발생

▣ 파이썬 제어문

2. if문

•if문의 형식

```
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

▣ 파이썬 제어문

2. if문

```
score = 90
if score >= 90:
    print 'Congratulations!!! '
```

```
Congratulations!!!
```

- score가 90과 같으니까 'True''로 Congratulations!!!'출력

```
a = 10
if a > 5:
   print 'Big'
else:
     print 'Small'
```

```
Big
```

- 만약 if문 조건이 만족되지 않으면 else문 statement 수행
- elif문이 없어도 바로 else문 삽입 가능

•statement가 1개인 경우 조건식과 한 줄에 위치 가능 (추천하지는 않음)

```
a = 10
if a > 5: print 'Big'
else: print 'Small'
```

```
Big
```

▣ 파이썬 제어문

2. if문

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

```
Negative
```

- 조건이 2가지로 각각의 경우에 따라 프린트 내용이 다르도록 코딩
- 두 조건 불만족 시 else문의 statement 수행

```
order = 'spagetti'

if order == 'spam':
   price = 500
elif order == 'ham':
   price = 700
elif order == 'egg':
   price = 300
elif order == 'spagetti':
   price = 900

print price
```

```
900
```

▣ 파이썬 제어문

2. if문

•위 코드와 동일한 dictionary 자료구조를 사용한 코드

```
order = 'spagetti'
menu = {'spam':500, 'ham':700, 'egg':300, 'spagetti':900}
price = menu[order]
print price
```

```
900
```

- 메뉴가 사전으로 구성되어 있음
- spam, ham, egg, spaghetti→ key
- 500, 700, 300, 900 → 각 key에 대응하는 value
- order가 spaghetti니까 spaghetti key에 value값이 price에 할당됨

▣ 파이썬 제어문

3. for문

•for문의 형식

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

- 컨테이너 객체에서 원소를 꺼내 타겟에 삽입
- statement는 타겟의 value를 활용하여 코딩

```
a = ['cat', 'cow', 'tiger']
for x in a:
    print len(x), x
```

```
3 cat
3 cow
5 tiger
```

- a는 리스트 → 리스트, 문자, 듀플, 사전은 콘테이너 객체
- len(x)의 x자리에 a 리스트의 첫번째 원소가 들어감
- len: 문자열의 길이를 구하는 내장함수

```
for x in [1,2,3]:
    print x,
```

```
1 2 3
```

▣ 파이썬 제어문

3. for문

```
print range(10)

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

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
0 1 2 3 4 5 6 7 8 9
```

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

```
sum = 0
for x in range(1, 11):
    sum = sum + x

print sum
```

```
55
```

- range(1, 11): 1은 start값, 11은 end 값
- range(1, 11) = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

```
prod = 1
for x in range(1, 11):
    prod = prod * x

print prod
```

```
3628800
```

- prod * x : prod라는 변수에 곱하기의 누적
- 1 * 2=결과 * 3 = 결과 * 4=……= 결과 * 10

▣ 파이썬 제어문

3. for문

•enumerate() 내장 함수: 컨테이너 객체가 지닌 각 요소값뿐만 아니라 인덱스 값도 함께 반환한다.

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

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

- k → enumerate하고 있는 컨테이너 객체의 index
- animal → 실제 값 들어옴
- 앞에 있는 0, 1, 2, 3이 그대로 각각의 문자열에 대한 index 역할 수행
- enumerate: key와 실제 그 값을 같이 가져옴
- key: 컨테이너 객체가 지니고 있는 index값

```
t = ('cat', 'dog', 'bird', 'pig')
for k, animal in enumerate(t):
    print k, animal
```

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

▣ 파이썬 제어문

3. for문

```
d = {'c':'cat', 'd':'dog', 'b':'bird', 'p':'pig'}
for k, key in enumerate(d):
    print k, key, d[key]
```

```
0 p pig
1 c cat
2 b bird
3 d dog
```

- dictionary 자체가 index를 관리하지 않아 index 바로 나오진 않음
- k에는 반드시 0부터 시작되는 index가 들어감
- k → 0, 1, 2, 3
- key → 하나의 문자로 이루어진 문자열(c, d, b, p)이 하나씩 들어감
- d[key]: 키워드 index
- d[c] → cat, d[d] → dog

▣ 파이썬 제어문

3. for문

•break: 루프를 빠져나간다.

```
for x in range(10):
    if x > 3:
        break
    print x

print 'done'
```

```
0
1
2
3
done
```

- break, continue
- range (10)→ [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
- break: 현재 수행중인 for문 또는 while문 반복 하지 않음
- x가 4가 되어 3보다 커지는 순간 for문 수행 중지 (break)

▣ 파이썬 제어문

3. for문

• continue: 루프 블록 내의 continue 이후 부분은 수행하지 않고 루프의 시작부분으로 이동한다.

```
for x in range(10):
    if x < 8:
        continue
    print x

print 'done'
```

```
8
9
done
```

- continue: 특정 조건이 만족되었을 때 continue하면서 하위 코드 무시
- x자리에 0~7까지는 continue가 수행되어 print X 무시

• else: 루프가 break에 의한 중단 없이 정상적으로 모두 수행되면 else 블록이 수행된다.

```
for x in range(10):
    print x,          # 콤마(,) 때문에 줄이 바뀌지 않는다.
else:
    print 'else block'

print 'done'
```

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

- ,(콤마)가 들어가 있어서 원소들이 바로 옆에 프린트 됨
- else black에는 콤마가 없어서 done은 아래 블록에 프린트

▣ 파이썬 제어문

3. for문

• break에 의하여 루프를 빠져나가면 else 블록도 수행되지 않는다.

```
for x in range(10):
    break
    print x,
else:
    print 'else block'

print 'done'
```

```
done
```

- 아무 조건 없는 break → 0부터 break 걸려 print X, else문 무시
- else 블록은 for문이 break 걸리지 않고 정상적 수행 시 수행 가능

▣ 파이썬 제어문

3. for문

• for 루프의 중첩

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

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

3 * 2 = 6
3 * 3 = 9
3 * 4 = 12
3 * 5 = 15
3 * 6 = 18
3 * 7 = 21
3 * 8 = 24
3 * 9 = 27
```

- 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

▣ 파이썬 제어문

4. While문

• while 조건식이 만족하는 동안 while 블록내의 statements 들을 반복 수행한다.

```
count = 1
while count < 11:
    print count
    count = count + 1
```

```
1
2
3
4
5
6
7
8
9
10
```

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

```
55
```

- a가 10이 되어 조건 불만족 시 중단되고 현재 누적 값 출력

▣ 파이썬 제어문

4. While문

```
x = 0
while x < 10:
    print x,          # 콤마(,) 때문에 줄이 바뀌지 않는다.
    x = x + 1
else:
    print 'else block'

print 'done'
```

```
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

파이썬 프로그래밍

제어문과 함수 기초

한국기술교육대학교
온라인평생교육원

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

• 함수의 장점
- 함수는 반복적인 코드를 없에 주어 코드의 길이를 짧게 만들어 준다.
- 코드의 유지보수를 쉽게 만들어 준다.

- 연관된 코드들을 여러 번 수행 가능하도록 함수에 엮음

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

print add(3, 4)

print add([1,2,3], [4,5,6])
```

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

- 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
- a와 b의 타입은 함수를 호출할 때 결정됨
- a 타입 → int, b의 타입 → int
- 인자를 받을 때 타입이 결정되기 때문에 동적인 특징을 가짐
- 7을 리턴 받아서 리턴받은 7의 값을 프린트
- [1, 2, 3] + [4, 5, 6] = [1, 2, 3, 4, 5, 6]

```
c = add(10, 30)
print c
```

```
40
```

- c = add(10, 30) = 40

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

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

```
f = add
print f(4, 5)

print f

print f is add
```

```
9
<function add at 0x10d8916e0>
True
```

- add → 하나의 식별자 (함수 식별자)
- 함수 = 객체 → 모든 변수 = 객체
- 1은 객체, a는 1이라는 객체를 가리키는 포인터(레퍼런스)
- add 변수 → 함수 객체를 가리키고 있는 레퍼런스
- f하고 add가 동일한 함수를 가리키고 있는 식별자
- <f 식별자의 형, 실제 식별자 이름, 객체가 놓여져 있는 메모리 위치>
- is 연산자: 일종의 예약어
- f와 add가 같은 객체인지, 같은 식별자인지 알아보는 키워드
- f is add = true

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

•함수의 몸체에는 최소한 한개 이상의 statement가 존재해야 함
그러므로, 아무런 내용이 없는 몸체를 지닌 함수를 만들 때에는 pass 라는 키워드를 몸체에 적어주어야 함

```
def simple():
    pass

print simple()
```

```
None
```

- simple 함수: 인자를 받는 것이 하나도 없음
- 내용을 안 적고 싶어도 반드시 적어야 함
- 비어 있는 경우 pass 예약어 사용
- pass 함수가 리턴하는 값이 없음
- simple 을 호출해서 받아내는 것이 없다. → none 객체 리턴

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

•함수에서 다른 함수 호출 가능

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

def myabs(x):
    if x < 0:
        x = -x
    return x

def addabs(a, b):
    c = add(a, b)
    return myabs(c)

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

```
12
```

- myabs는 인자를 x로 받아, x가 0보다 작으면 부호를 바꿔줌
- c = add(-5, -7) = -5 + -7 = -12
- myabs(-12) = 12

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

•인자의 이름과 함께 인자 값을 넘겨줄 수 있다.

```
def minus(a, b):
    return a - b
print minus(a=12, b=20)
print minus(b=20, a=12)
```

```
-8
-8
```

- def 함수이름(식별자) 인자 :
- def minus(12, 20)
- 인자의 이름을 적어주면 반드시 a값 먼저 안 적어도 괜찮음

•인자의 디폴트 값을 지정할 수 있다.

```
def incr(x, y=1):
    return x + y

print incr(5)

print incr(5, 10)
```

```
6
15
```

- y=1 → y의 디폴트 값이 1이다.
- 디폴트 값이 정해져 있으면 인자를 하나만 줘도 괜찮음
- 5 + 1 = 6

▣ 파이썬 함수 기초

1. 함수의 장점 및 함수 사용법

•두 개 이상의 값을 동시에 반환할 수 있다.

```
def calc(x, y):
    return x + y, x - y, x * y, x / y

print calc(10, 2)
```

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

- 인자는 2개인데, 총 4개(더하기, 빼기, 곱하기, 나누기한 값)을 리턴
- 여러 개의 값이 콤마의 형태로 묶여있는 자료: tuple(튜플)

▣ 파이썬 함수 기초

2. 함수 호출시 동적인 자료형 결정

•파이썬에서는 모든 객체는 동적으로 (실행시간에) 그 타입이 결정된다.
- 함수 인자는 함수가 호출되는 순간 해당 인자에 전달되는 객체에 따라 그 타입이 결정된다.
- 함수 몸체 내에서 사용되는 여러 가지 연산자들은 함수 호출시에 결정된 객체 타입에 맞게 실행된다.

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

c = add(1, 3.4)
d = add('dynamic', 'typing')
e = add(['list'], ['and', 'list'])
print c
print d
print e
```

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

- add(a, b) → a는 정수, b는 실수
- 정수 + 실수 = 실수
- 문자열 + 문자열 = 문자열
- 리스트 + 리스트 = 리스트

▣ 파이썬 함수 기초

3. 재귀적 함수 호출

•재귀(Recursive) 함수: 함수 몸체에서 자기 자신을 호출하는 함수
 - 수학에서 점화식과 유사한 코드
 - 반드시 종결 조건 및 종결 조건이 만족할 때의 반환 값이 있어야 한다.
•1부터 N까지 더하는 재귀 함수

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

print sum(10)
```

```
55
```

- return 10 + sum(10-1)
- return 10 + 9 + 8+ 7+ 6+ 5+ 4+ 3+ 2+ 1 = 55
- if절이 종결되었을 때 (만족되었을 때) return문이 함께 보여야 함