[CSE1017]
프로그래밍기초

Control Structures

#03. 제어 구조

김현하

한양대학교 ERICA 소프트웨어학부

2024년도 1학기

python™

목차

논리식

논리식과 논리값

- 논리식logical expression, Boolean expression
  - 계산 결과가 참`True` 또는 거짓`False`으로만 나타나는 식
- 논리값logical value, Boolean value
  - 논리식의 계산 결과

```
>>> True
>>> False
>>> true
```

논리 연산

논리연산자logical operator, Boolean operator

연산	논리곱	논리합	논리역
기호	and	or	not

우선순위precedence

우선순위	연산자
가장 높음	not
높음	and
낮음	or

진리표truth table

and	True	False
True	True	False
False	False	False

or	True	False
True	True	True
False	True	False

not	-
True	False
False	True

단축 계산short-circuit evaluation

- 수식의 이항연산은 양쪽 피연산자의 계산 결과가 모두 있어야 함
- 논리 연산은 경우에 따라 단축 계산이 가능함
  - and : 왼쪽 피연산자의 결과가 False 이면 오른쪽 피연산자의 계산을 생략하고 바로 False
  - or : 왼쪽 피연산자의 계산 결과가 True 이면 오른쪽 피연산자의 계산을 생략하고 바로 True

```
>>> def loop(): loop()
>>> loop()
```

비교 논리식

- 비교연산 : 두 값을 비교하는 계산
  - **수**는 크기를 기준으로 비교한다.
  - **논리값**은 본질적으로는 순서가 없지만, Python 언어에서는 내부적으로 `True`를 1로, `False`를 0으로 처리하므로 `True`가 `False`보다 크다.
  - **문자열**은 각 문자에 할당된 ASCII 값의 크기를 기준으로 **사전 순**(사전에서 단어를 나열하는 순서)으로 비교한다.

비교 연산자 기호

이항연산자binary operator, 중위표기infix notation

연산	작음	큼	작거나 같음	크거나 같음	같음	같지 않음
기호	<	>	<=	>=	==	!=

비교연산자 체이닝comparison operator chaining

$e_0\ op_1\ e_1\ op_2\ e_2\ \ldots\ e_{n-1}\ op_n\ e_n$ = $e_0\ op_1\ e_1$ and $e_1\ op_2\ e_2$ and ... and $e_{n-1}\ op_n\ e_n$

```
x < y < z ·····▶ x < y and y < z
```

조건문

조건문conditional

areacircle.py

```
1  from math import pi
2  def area_circle(radius, n):
3      area = pi * radius ** 2
4      return round(area, n)
5
6  print(area_circle(3, 1))
7  print(area_circle(-3, 1)) # 반지름이 음수인 경우?
```

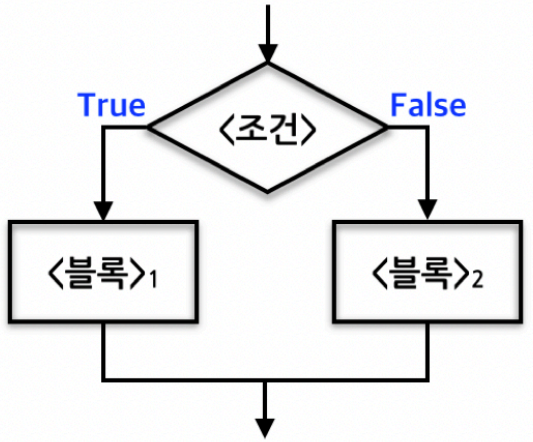
조건문의 문법과 의미

문법

```
if <조건>:
    <블록>1
else:
    <블록>2
```

의미

- <조건>의 계산 결과가
  - True 이면 if 에 달린 <블록>1을 실행하고
  - False 이면 else 에 달린 <블록>2를 실행한다.
- 어떤 상황에도 <블록>1과 <블록>2 중 하나만 실행한다.



- 〈조건〉 : 논리식
- 〈블록〉 : 동일하게 들여쓴 한 줄 이상의 코드로 구성된 블록

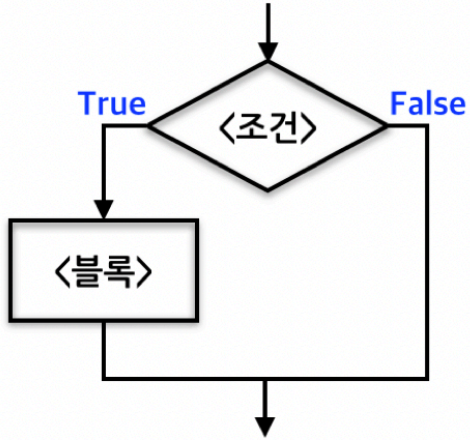
조건문의 문법과 의미

문법

```
if <조건>:
    <블록>
```

의미

- <조건>의 계산 결과가
  - True 이면 if 에 달린 <블록>을 실행하고
  - False 이면 아무 일도 하지 않고 지나간다.



- <조건> : 논리식
- <블록> : 동일하게 들여쓴 한 줄 이상의 코드로 구성된 블록

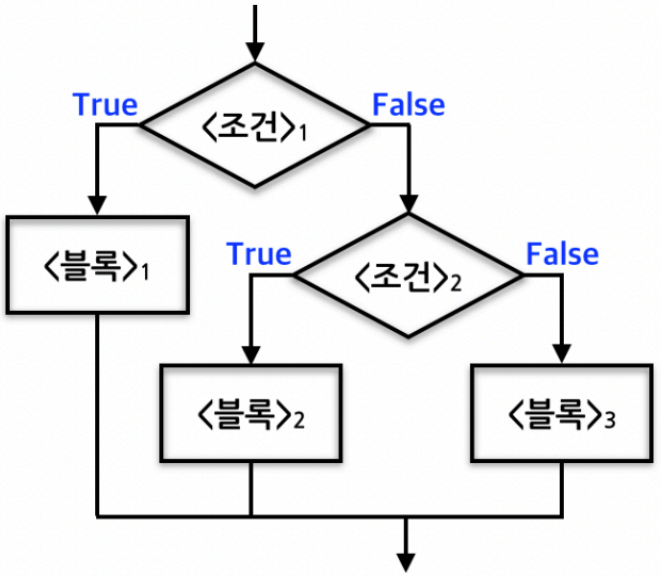
조건문의 문법과 의미

문법

```
if <조건>1:
    <블록>1
elif <조건>2:
    <블록>2
else:
    <블록>3
```

의미

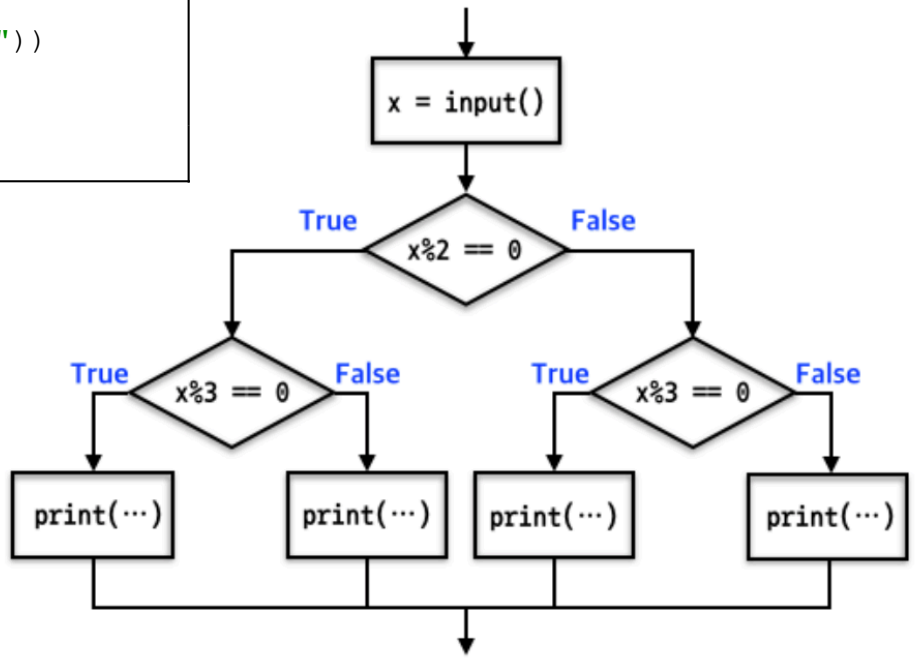
- <조건>$_1$의 계산 결과가 True이면 if 에 달린 <블록>$_1$을 실행하고,
- <조건>$_1$의 계산 결과가 False이고 <조건>$_2$의 계산 결과가 True이면 <블록>$_2$를 실행하고,
- <조건>$_2$의 계산 결과도 False이면 <블록>$_3$을 실행한다.
- 어떤 상황에도 <블록>$_1$, <블록>$_2$, <블록>$_3$ 중 하나만 실행한다.



- 〈조건〉 : 논리식
- 〈블록〉 : 동일하게 들여쓴 한 줄 이상의 코드로 구성된 블록

중첩된nested 조건문

```
1  x = int(input("Enter your number: "))
2  if x % 2 == 0:
3      if x % 3 == 0:
4          print(x, "is divisible by 2 and 3"))
5      else:
6          print(x, "is divisible by 2 but not by 3"))
7  else:
8      if x % 3 == 0:
9          print(x, "is divisible by 3 but not by 2"))
10     else:
11         print(x, "is not divisible by 2 or 3"))
```



반복문

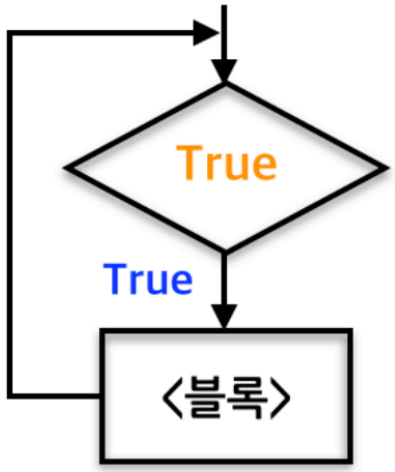
무조건 반복

```
1  from math import pi
2  def area_circle(radius, n):
3      if radius > 0:
4          area = pi * radius ** 2
5          return round(area, n)
6      else:
7          return 0.0
8
9  print("Circle Area Calculator")
10 r = input("Radius? ")
11 p = input("Precision? ")
12 area = area_circle(int(r), int(p))
13 print("The area of a circle with radius", r, "is", area, ".")
```

무조건 반복

(무조건) 반복문

```
while True:
    <블록>
```



무조건 반복

```
1  import math
2  def area_circle(radius, n):
3      if radius > 0:
4          area = pi * radius ** 2
5          return round(area, n)
6      else:
7          return 0.0
8
9  print("Circle Area Calculator")
10 while True:
11     r = input("Radius? ")
12     p = input("Precision? ")
13     area = area_circle(int(r), int(p))
14     print("The area of a circle with radius", r, "is", area, ".")
15 print("Please come back again.")
```

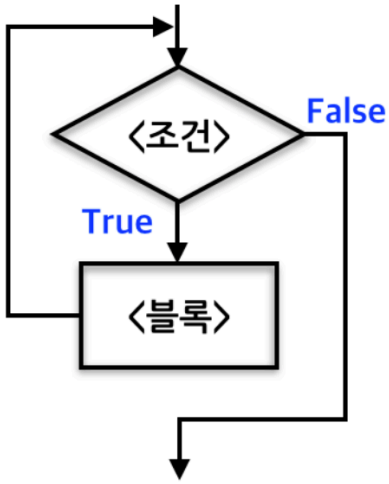
while 루프의 문법과 의미

문법

```
while <조건>:
    <블록>
```

의미

- <조건>의 계산 결과가 True인 동안 <블록>을 반복 실행하다가
- <조건>의 계산 결과가 False가 되면 루프를 빠져나간다.



- 〈조건〉 : 논리식
- 〈블록〉 : 동일하게 들여쓴 한 줄 이상의 코드로 구성된 블록

조건 반복

```
1  import math
2  def area_circle(radius, n):
3      if radius > 0:
4          area = pi * radius ** 2
5          return round(area, n)
6      else:
7          return 0.0
8
9  print("Circle Area Calculator")
10 more = 'y'
11 while more == 'y':
12     r = input("Radius? ")
13     p = input("Precision? ")
14     area = area_circle(int(r), int(p))
15     print("The area of a circle with radius", r, "is", area, ".")
16     more = input("Press y to continue, any other key to exit. ")
17 print("Please come back again.")
```

입력 확인input validation

- `isdigit()` : 문자열이 숫자(digit)로만 구성되어 있는지 검사하는 함수
- 메소드method : <문자열>`.isdigit()` 와 같은 형태의 함수
  - 클래스class는 객체object라고 하는 메모리에 거주할 실체instance 데이터를 찍어내는 형판(틀)이다.
  - 클래스라는 형판으로 객체를 얼마든지 만들어 낼 수 있다.
  - 메소드는 클래스가 지닌 함수로, 객체에는 이 함수를 실행할 수 있는 기능이 저절로 장착된다.
- **<문자열>**`.isdigit()`
  - **<문자열>**은 객체이다.
  - **<문자열>** 객체는 `str` 클래스로 찍어낸다.
  - `isdigit()`은 `str` 클래스가 지니고 있는 메소드 이다.
  - **<문자열>**`.isdigit()`와 같은 형식으로 **<문자열>** 객체에게 `isdigit()` 호출 메시지를 보내면, 자신을 대상으로 `isdigit()`을 실행하여 결과를 리턴한다.

입력 확인 코드 패턴

코드 패턴

```
x = input()
while not <통과조건>:
    x = input()
```

의미

- <통과조건>은 통과시킬 우량 입력 조건을 문자열 x를 기준으로 작성한 논리식이다.
- <통과조건>을 만족하지 못하면, while 루프의 조건이 True가 되어 재입력을 받는다.
- <통과조건>을 만족할 때까지 이 과정을 반복하는데, <통과조건>을 만족하는 순간, while 루프의 조건이 False가 되어 루프를 빠져나가 입력 확인 과정을 통과한다.

```
1  x = input()
2  while not x.isdigit():
3      x = input()
```

```
1  import math
2  def area_circle(radius, n):
3      if radius > 0:
4          area = pi * radius ** 2
5          return round(area, n)
6      else:
7          return 0.0
8
9  print("Circle Area Calculator")
10 more = 'y'
11 while more == 'y':
12     r = input("Radius? ")
13     while not r.isdigit():
14         r = input("Radius? ")
15     p = input("Precision? ")
16     while not p.isdigit():
17         p = input("Precision? ")
18     area = area_circle(int(r), int(p))
19     print("The area of a circle with radius", r, "is", area, ".")
20     more = input("Press y to continue, any other key to exit. ")
21 print("Please come back again.")
```

문자열 해부

인덱스index

- 문자열은 문자를 일렬로 나열한 일차원 벡터vector로 볼 수 있다.
- 문자열의 각 문자는 인덱스라고 하는 고유의 위치번호가 매겨진다.
- 각 문자의 인덱스는 맨 왼쪽 문자부터 0으로 시작해서 오른쪽으로 가면서 1씩 증가한다.
- 각 문자의 인덱스는 맨 오른쪽 문자부터 -1로 시작해서 왼쪽으로 가면서 1씩 감소한다.

	0	1	2	3	4	
"	컴	퓨	터	과	학	"
	-5	-4	-3	-2	-1	

- 문자열의 길이가 n이면 인덱스의 범위는 0 ~ n-1 과 -n ~ -1이다.
- 범위에 벗어나는 인덱스를 사용하면 `IndexError`가 발생한다.

문자열 분해

문법

<문자열>[<정수식>]

의미

- <문자열>을 s 라고 하고
- <정수식>을 계산한 결과 값을 i 라고 하면,
- s의 인덱스 i 에 위치하는 문자를 내어준다.

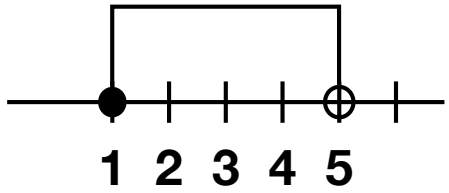
```
>>> "컴퓨터과학"[0]
>>> "컴퓨터과학"[2]
```

```
>>> cs = "컴퓨터과학"
>>> cs[4]
>>> cs[-4]
>>> cs[-6]
>>> cs[1.0]
>>> cs[1 + 1]
```

문자열 분해

문법

<문자열>[<정수식>$_1$:<정수식>$_2$]



의미

- <문자열>을 s 라고 하고,
- <정수식>$_1$을 계산한 정수를 begin 이라고 하고,
- <정수식>$_2$를 계산한 정수를 end 라고 하면,
- 문자열 s에서 begin 인덱스와 end 인덱스 사이의 문자를 포함하는 문자열을 새로 만들어 준다(s[begin]은 포함, s[end]는 제외).
- <정수식>$_1$이 주어지지 않으면 기본값은 0 이고, <정수식>$_2$가 주어지지 않으면 기본값은 len(s) 이다. s[:]는 s와 동일한 문자열을 복제한다.

```
>>> cs = "컴퓨터과학"
>>> cs[3:5]
>>> cs[3:]
>>> cs[:2]
>>> cs[:]
>>> cs[:3] + '공' + cs[4:]
```

문자열 메소드 partition

문법

```
<문자열>.partition(<기준문자열>)
```

의미

- partition 메소드는 <기준문자열>을 중심으로 문자열을 세 조각으로 나누어 준다.
- (소괄호로 감싸고생략가능) 쉼표로 구분한 조각을 튜플tuple이라고 한다.
- 만약 <기준문자열>이 <문자열>에 없으면 첫째 조각이 문자열 전체가 되고 둘째와 셋째 조각은 빈 문자열인 튜플을 내준다.
- 튜플도 문자열과 같이 인덱스로 접근할 수 있다.

```
>>> pi = "3.14159".partition(".")
>>> pi[0]
>>> pi[1]
>>> pi[2]
>>> pi[3]
```