파이썬 프로그래밍

한빛아카데미 Hanbit Academy, Inc.

Chapter 06
반복문

목차

Preview


for문
기초 다지기
반복문이 필요한 이유
for문의 개념
활용하기
for문으로 합계 구하기
중첩 for문
while문
for문
vs
while문
무한 반복
break문
continue문

학습목표

- 반복문을 사용하는 이유를 파악합니다.
- for문의 문법을 익히고, 활용하여 코딩합니다.
- while문과 for문의 차이를 이해하고, 무한 반복을 이해합니다.
- break, continue와 같은 제어문의 사용법을 익힙니다

Section 01
for문 기초 다지기

반복문이 필요한 이유

- 반복문
  - 특정 부분을 원하는 횟수만큼 반복하는 구문

- 반복문을 사용하는 이유
  - 수백 줄의 코드를 단 몇 줄로 줄이는 효율을 발휘함
  - 다음 결과를 출력하는 프로그램 만들어 보기

```
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
```

반복문이 필요한 이유

- 반복문을 사용하는 이유
  - 방법1) 프린트 구문을 세 번 작성하기

```
print("난생처음 파이썬은 재미있습니다. ^^")
print("난생처음 파이썬은 재미있습니다. ^^")
print("난생처음 파이썬은 재미있습니다. ^^")
```

  - 방법2) 반복문을 사용하기

```
for i in range(3) :
    print("난생처음 파이썬은 재미있습니다. ^^")
```

for문의 개념

for문의 형식

```
for 변수 in range( 시작값, 끝값+1, 증가값 ) :
    반복할 문장
```

그림 6-1 for문의 형식

- range(시작값, 끝값+1, 증가값) : 지정된 범위의 값을 반환함
- range(0, 3, 1) : 0에서 시작해서 2까지 1씩 증가하는 값들을 반환함
  - 증가값을 생략할 경우 1로 인식함
  - 따라서 range(0, 3, 1)은 range(0, 3)과 동일함
  - 시작값이 0이기 때문에 시작값도 생략 가능함
  - 따라서 range(3)만 써도 range(0, 3, 1)과 동일하게 인식함

for문의 개념

- range() 함수를 사용한 코드

```
>>> for i in range(0, 3, 1) :
        print("난생처음 파이썬은 재미있습니다. ^^")
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
```

  - range(0, 3, 1)은 [0, 1, 2]로 대체될 수 있음

```
>>> for i in [0, 1, 2] :
        print("난생처음 파이썬은 재미있습니다. ^^")
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
난생처음 파이썬은 재미있습니다. ^^
```

for문의 개념

- for i in [0, 1, 2]의 작동
  - i에 0, 1, 2를 차례로 대입하면서 3회 반복함
    - 1회: i에 0을 대입한 후 print() 함수 수행
    - 2회: i에 1을 대입한 후 print() 함수 수행
    - 3회: i에 2를 대입한 후 print() 함수 수행

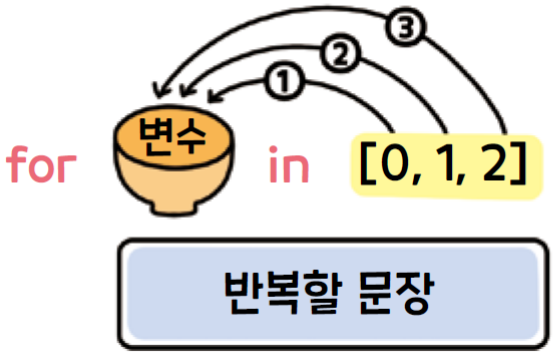


그림 6-2 for i in [0, 1, 2]의 작동

[0, 1, 2]는 리스트입니다. 리스트에 대해서는 7장에서 자세히 다룹니다. 지금은 0, 1, 2가 한꺼번에 저장된 것 정도로 기억하세요.

for문의 개념

- i 값에 접근하기
  - 각 행의 맨 앞에 i 값을 출력하기

```
>>> for i in range(0, 3, 1) :
        print(i, “: 난생처음 파이썬은 재미있습니다. ^^")
0 : 난생처음 파이썬은 재미있습니다. ^^
1 : 난생처음 파이썬은 재미있습니다. ^^
2 : 난생처음 파이썬은 재미있습니다. ^^
```

  - 1~10까지 숫자들을 차례대로 출력하기

```
>>> for i in range(1, 11, 1) :
        print(i , end='')
1 2 3 4 5 6 7 8 9 10
```

for문의 개념

확인문제

다음과 같은 결과가 나올 수 있도록, 코드의 빈칸을 채우시오.

```
for i in range          :
    print(i , end='')
```

```
2  3  4  5
```

정답

Click!

LAB 학생 줄 세우기

A, B, C, D, E 다섯 명의 학생들에게 도시락을 나눠주려고 합니다. 다섯 명의 학생들이 도시락을 받기 위해 순서대로 줄을 서려면 총 몇 가지의 경우의 수가 나올까요?

이런 경우 팩토리얼 함수를 사용하면 쉽게 해결됩니다. 팩토리얼(Factorial)은 1부터 n까지 숫자의 곱을 의미합니다. 팩토리얼은 기호로 !를 사용합니다. 예로 5!은 1×2×3×4×5로 계산되어 120이 됩니다.

실행 결과

```
A, B, C, D, E 학생들을 순서대로 세우는 경우의 수 : 120
```

LAB 학생 줄 세우기

1. lab06-01.py 파일을 만들고, A~E 친구들이 줄을 설 수 있는 경우의 수를 계산하기 위해 필요한 변수를 선언하기
   - 이때 팩토리얼을 저장할 변수 fact의 초기 값을 1로 설정 해야 함.
     - 초기 값을 0으로 하면 0과의 모든 곱셈이 0이 되기 때문

```
i = 0
fact = 1
friends_num = 5
```

2. for문을 활용해 i가 1부터 5까지 순서대로 커지며 fact에 곱하기
   - 끝값을 '입력한값+1'로 해야 입력한 값까지 곱셈이 됨

```
for i in range(1, friends_num+1, 1):
    fact = fact * i

print("A, B, C, D, E 학생들을 순서대로 세우는 경우의 수:", fact)
```

3. <Ctrl>+<S>를 눌러 저장한 후, <F5>를 눌러 실행 결과 확인하기
   - friends_num의 숫자를 변경하며 팩토리얼 계산이 잘 되는지 추가로 확인 해보기

Section 02
for문 활용하기

for문을 활용하여 합계 구하기

for문을 활용한 반복적인 덧셈

① 한글로 코드 준비하기

```
1부터 10까지 변할 변수 i 준비

for 변수 i가 1을 시작으로 10까지 1씩 증가 :
    hap 값에 i 값을 더해 줌

hap의 값을 출력
```

하나 더 알기 **의사 코드**

프로그램 코드가 아닌, 코드와 비슷하게 글로 적는 것을 '의사 코드(pseudocode, 슈도 코드)'라고 부릅니다. 의사 코드는 진짜로 작동하지는 않지만, 흐름을 파악하는 데 도움이 됩니다. 또한 파이썬뿐 아니라 다른 프로그래밍 언어로 변환하기도 좋습니다. 컴퓨터 분야에서 종종 나오는 용어이므로 기억해 두기 바랍니다.

for문을 활용하여 합계 구하기

- for문을 활용한 반복적인 덧셈

② 파이썬 코드로 변환하기

[코드 6-2]

```
i = 0

for i in range(1, 11, 1) :
    hap = hap + i

print("1에서 10까지의 합 : ", hap)
```

[실행결과]

```
Traceback (most recent call last):
    File "C:/FirstPython/Chapter06/06-01.py", line 4, in <module>
        hap = hap + i
NameError: name 'hap' is not defined
```

for문을 활용하여 합계 구하기

- for문을 활용한 반복적인 덧셈
  - [코드 6-2]에서 오류가 발행한 이유
    - 4행에서 변수 hap을 선언하지 않았기 때문
    - 즉 hap에 어떤 값이 있어야만 다시 hap 자기 자신에 누적할 수 있음

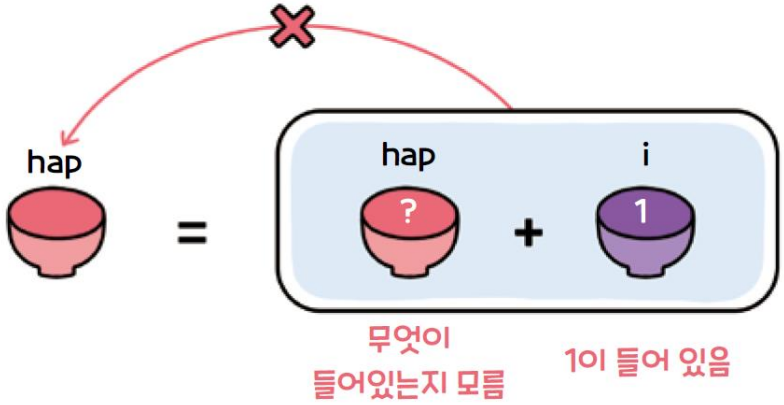


그림 6-3 hap 변수 안에 무엇이 들어있는지 모름

for문을 활용하여 합계 구하기

- for문을 활용한 반복적인 덧셈
  - [코드 6-2] 수정

[코드 6-3]

```
i = 0
hap = 0

for i in range(1, 11, 1) :
    hap = hap + i

print("1에서 10까지의 합 : ", hap)
```

[실행결과]

```
1에서 10까지의 합 : 55
```

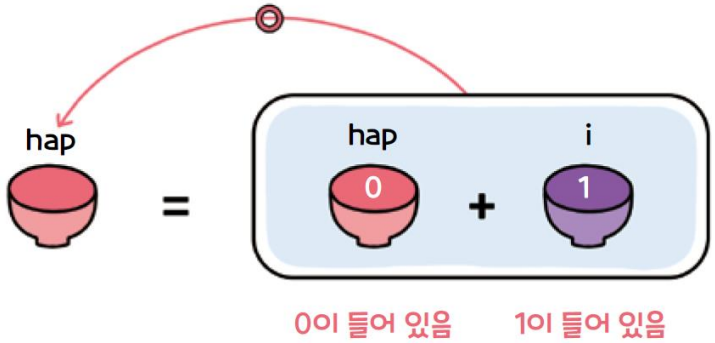


그림 6-4 hap 변수에 뭐가 있는지 알고 있음

for문을 활용하여 합계 구하기

- for문을 활용한 반복적인 덧셈
  - [코드 6-3]에서 변수 i와 hap의 값 변화

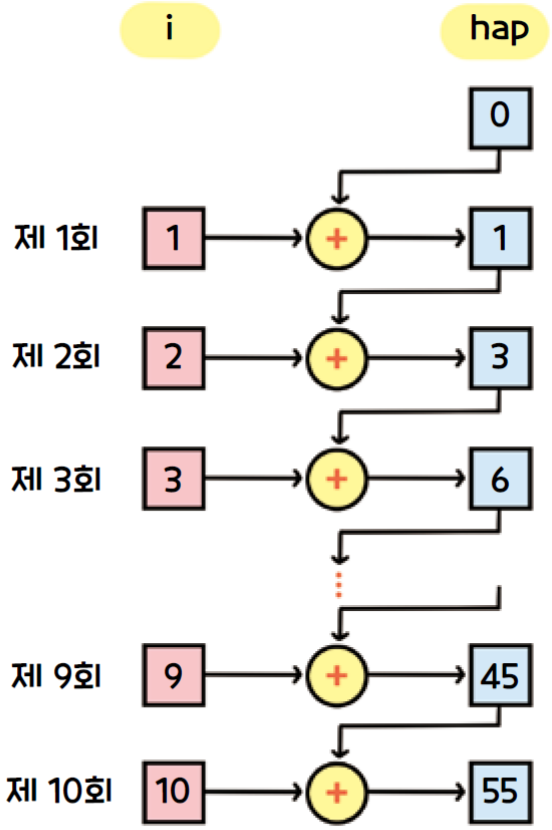


그림 6-5 변수 i와 hap의 변화

for문을 활용하여 합계 구하기

- for문을 활용한 반복적인 덧셈
  - 1000~2000 사이에서 홀수의 합을 구하는 프로그램

[코드 6-4]

```
i, hap = 0, 0

for i in range(1001, 2001, 2) :
    hap += i

print("1000에서 2000까지의 홀수의 합 :", hap)
```

[실행결과]

```
1000에서 2000까지의 홀수의 합 : 750000
```

for문을 활용하여 합계 구하기

확인문제

다음은 500부터 1000까지의 짝수의 합을 구하는 코드이다. 코드의 빈칸을 채우시오.

```
i, hap = 0, 0
for i in range [          ]:
    hap += i
print("500에서 1000까지의 짝수의 합 :", hap)
```

```
500에서 1000까지의 짝수의 합 : 188250
```

정답

Click!

중첩 for문

중첩 for문

- for문 안에 또 for문을 사용할 수도 있음
- 이를 중첩 for문 또는 중복 for문이라 함
- 중첩 for문의 실행 횟수: 바깥 for문 반복 횟수 × 안쪽 for문 반복 횟수

```
>>> for i in range (3) :
        for k in range(2) :
            print("난생처음은 쉽습니다. ^^ (i값:", i ,", k값:", k,")")
난생처음은 쉽습니다. ^^ (i값: 0, k값: 0)
난생처음은 쉽습니다. ^^ (i값: 0, k값: 1)
난생처음은 쉽습니다. ^^ (i값: 1, k값: 0)
난생처음은 쉽습니다. ^^ (i값: 1, k값: 1)
난생처음은 쉽습니다. ^^ (i값: 2, k값: 0)
난생처음은 쉽습니다. ^^ (i값: 2, k값: 1)
```

그림 6-6 중첩 for문의 동작 개념

중첩 for문

중첩 for문의 처리되는 순서

1) 외부 for문 1회: i에 0을 대입
   - 내부 for문 1회: k에 0을 대입한 후에 print() 수행
   - 내부 for문 2회: k에 1을 대입한 후에 print() 수행
2) 외부 for문 2회: i에 1을 대입
   - 내부 for문 1회: k에 0을 대입한 후에 print() 수행
   - 내부 for문 2회: k에 1을 대입한 후에 print() 수행
3) 외부 for문 3회: i에 2를 대입
   - 내부 for문 1회: k에 0을 대입한 후에 print() 수행
   - 내부 for문 2회: k에 1을 대입한 후에 print() 수행

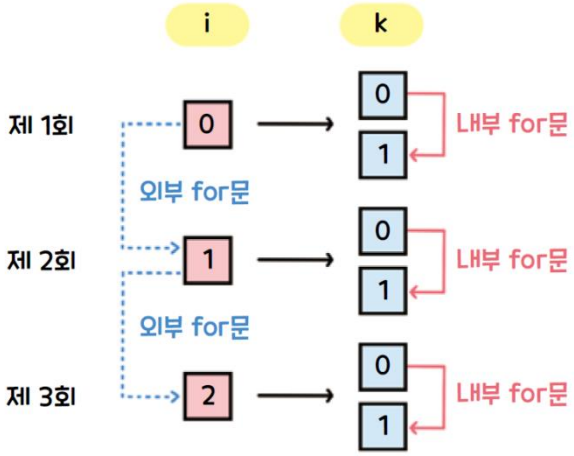


그림 6-7 중첩 for문에서 i와 k 값의 변화

중첩 for문

확인문제

다음 코드는 총 몇 번 출력되는가?

```
for i in range (2) :
    for k in range(3) :
        print("중첩 for문입니다")
```

정답

Click!

LAB 구구단 계산기 만들기

2단부터 9단까지 구구단을 출력하는 구구단 계산기를 만들어 봅시다.

2단~9단까지 for문을 사용해야 합니다. 그런데 2단에서도 다시 2×1, 2×2, 2×3, ..., 2×9까지 곱하는 숫자가 1부터 9로 변경되기 때문에 곱하는 숫자도 역시 for문을 사용해야 합니다. 결국 바깥 for문은 2단, 3단, ..., 9단을, 안쪽 for문은 1, 2, 3...으로 곱하는 숫자를 사용해야 합니다. 즉, 중첩 for문을 사용해서 코드를 작성해야 합니다.

실행 결과

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

구구단의 생성 규칙

X	1	2	3	4	5	6	7	8	9	10
1	1	2	3	4	5	6	7	8	9	10
2	2	4	6	8	10	12	14	16	18	20
3	3	6	9	12	15	18	21	24	27	30
4	4	8	12	16	20	24	28	32	36	40
5	5	10	15	20	25	30	35	40	45	50
6	6	12	18	24	30	36	42	48	54	60
7	7	14	21	28	35	42	49	56	63	70
8	8	16	24	32	40	48	56	64	72	80
9	9	18	27	36	45	54	63	72	81	90
10	10	20	30	40	50	60	70	80	90	100

LAB 구구단 계산기 만들기

1. lab06-02.py 파일을 만들고, 1행에서 단으로 사용할 변수 i와 곱할 숫자로 사용할 k를 준비하기

```
i = 0
k = 0
```

2. 바깥쪽 for문은 2단~9단까지 8번 반복하며, 안쪽 for문은 1~9까지 9번 반복함
   - 안쪽 for문에서 i와 k를 곱하여 구구단을 출력

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

3. <Ctrl>+<S>를 눌러 저장한 후, <F5>를 눌러 실행 결과 확인하기

Section 03
while문이란?

for문과 while문의 비교

- for문 vs while문
  - for문: 반복할 횟수를 range()에서 결정한 후에, 그 횟수만큼 반복함
  - while문: 반복 횟수를 결정하기 보다는 조건식이 참인 경우에 반복함

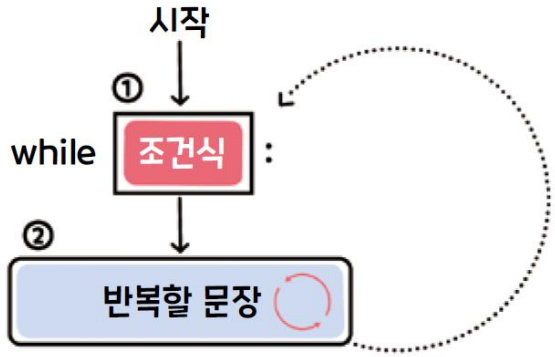


그림 6-8 while문의 형식과 실행 순서

for문과 while문의 비교

- for문 vs while문
  - "난생처음 ~" 문장을 3번을 반복하는 for문

```
>>> for i in range(0, 3, 1) :
        print(i, ": 난생처음 파이썬은 재미있습니다. ^^")
```

  - while문으로 변경하기
    - 강제로 1을 증가시키는 행을 추가해야 함

```
>>> i=0
>>> while (i < 3) :
        print(i" : 난생처음 파이썬은 재미있습니다. ^^")
        i = i + 1
0 : 난생처음 파이썬은 재미있습니다. ^^
1 : 난생처음 파이썬은 재미있습니다. ^^
2 : 난생처음 파이썬은 재미있습니다. ^^
```

for문과 while문의 비교

확인문제

다음 코드를 실행하면 몇 번 출력되는가?

```
i=0
while (i<5) :
    print("즐거운 난생 처음 ^^")
    i = i+1
```

정답

Click!

while문의 무한 반복

- 무한 루프(무한 반복)
  - 반복문을 빠져나올 조건이 없어 무한히 while문 내부를 반복함
  - 'while 조건식 : '의 조건식을 True로 지정하면 무한 루프로 지정됨

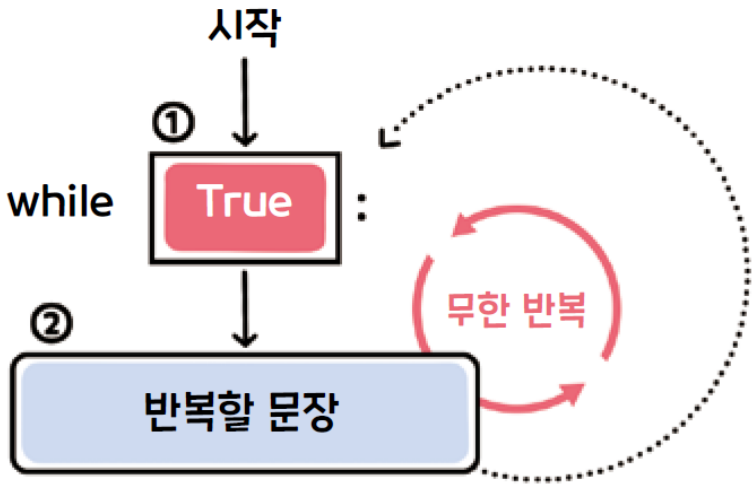


그림 6-9 while을 이용한 무한 루프

while문의 무한 반복

- 무한 반복하는 간단한 while문
  - 글자 'ㅎ' 무한 출력하기
    - 무한 반복 중지하는 방법 : <Ctrl> + <C> 입력

```
>>> while True :
        print("ㅎ", end = " ")
ㅎ ㅎ ㅎ ㅎ ㅎ ㅎ ...무한 반복...
```

while문의 무한 반복

무한 반복하는 간단한 while문

- 입력한 두 숫자의 합계를 반복해서 계산하는 프로그램
  - 사용자가 프로그램을 중단하기 위해 <Ctrl>+<C>를 누르기 전까지는 계속 두 숫자를 입력받고 두 수를 더한 결과를 출력함

[코드 6-5]

```
hap = 0
num1, num2 = 0, 0

while True :
    num1 = int(input("숫자1 ==> "))
    num2 = int(input("숫자2 ==> "))

    hap = num1 + num2
    print(num1, "+", num2, "=", hap)
```

[실행결과]

```
숫자1 ==> 30
숫자2 ==> 77
30 + 77 = 107

숫자1 ==> 8
숫자2 ==> 12345
8 + 12345 = 12353

...무한 반복 Ctrl+C로 종료...
```

사용자 입력

사용자 입력

break문과 continue문

- 반복문을 탈출하는 break문
  - 반복문 안에서 break문을 만나면 무조건 반복문을 빠져나옴

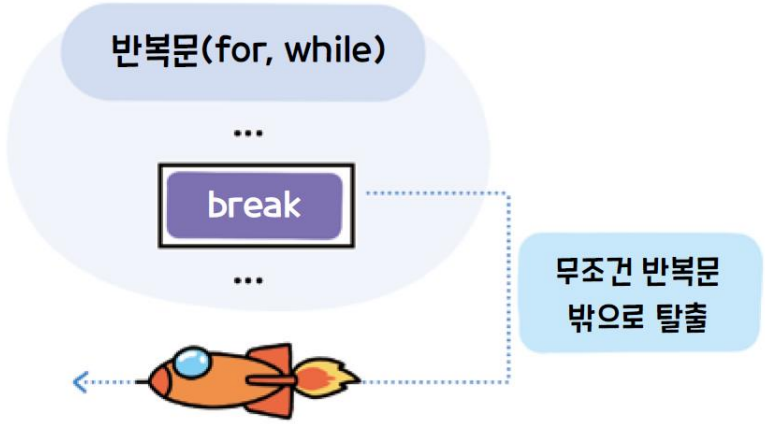


그림 6-10 break문의 작동

```
>>> for i in range(1, 1001, 1) :
        print("반복문을", i, "회 실행합니다.")
        break
반복문을 1 회 실행합니다.
```

break문과 continue문

반복문을 탈출하는 break문

- [코드 6-5]를 break문을 사용하여 첫 번째 숫자(num1)에 0을 입력하면 반복문을 빠져나가도록 수정하기

[코드 6-6]

```
hap = 0
num1, num2 = 0, 0

while True :
    num1 = int(input("숫자1 ==> "))
    if num1 == 0 :
        break
    num2 = int(input("숫자2 ==> "))

    hap = num1 + num2
    print(num1, "+", num2, "=", hap)

print("0을 입력해서 계산을 종료합니다.")
```

[실행결과]

```
숫자1 ==> 30
숫자2 ==> 77
30 + 77 = 107

숫자1 ==> 0
0을 입력해서 계산을 종료합니다.
```

사용자 입력

사용자 입력

break문과 continue문

- 처음으로 돌아가는 continue문
  - break문은 반복문을 빠져나가지만, 이와 달리 continue문은 남은 부분을 모두 건너뛰고, 반복문의 처음으로 돌아감

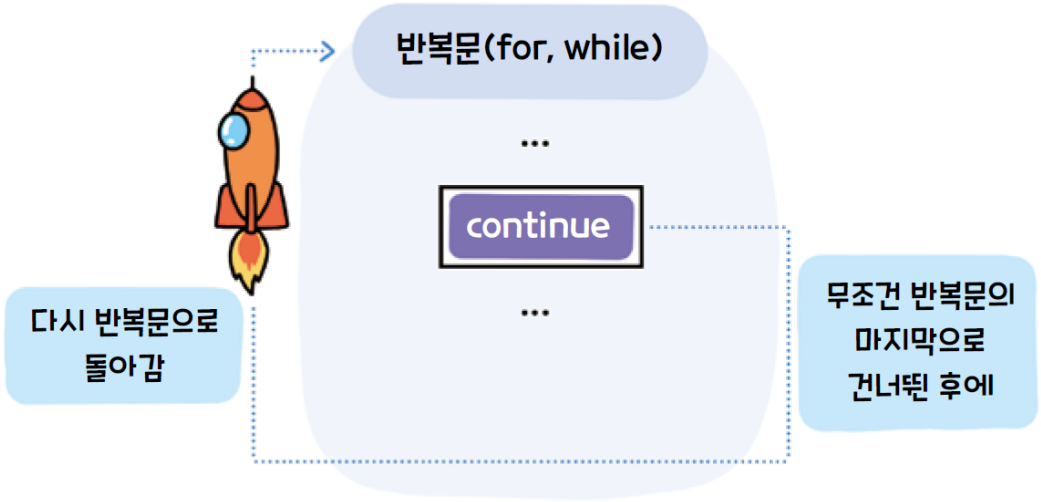


그림 6-11 continue문의 작동

break문과 continue문

처음으로 돌아가는 continue문

- 숫자 1~100을 더하되 4의 배수는 더하지 않는 프로그램
  - 즉 1+2+3+5+6+7+9+...와 같이 4의 배수를 건너뛰고 합계를 구함

[코드 6-7]

```
i, hap = 0, 0

for i in range(1,101,1) :
    if i % 4 == 0 :
        continue

    hap += i

print("1~100의 합계(4의 배수 제외) : ", hap)
```

[실행결과]

```
1~100의 합계(4의 배수 제외) : 3750
```

break문과 continue문

- 처음으로 돌아가는 continue문
  - [코드 6-7]의 동작 단계
    - 제1회: i 값 1을 4로 나누면 나머지는 1(거짓) → hap += 1 수행
    - 제2회: i 값 2를 4로 나누면 나머지는 2(거짓) → hap += 2 수행
    - 제3회: i 값 3을 4로 나누면 나머지는 3(거짓) → hap += 3 수행
    - **제4회: i 값 4를 4로 나누면 나머지는 0(참) → continue 수행**
      다시 3행으로 돌아가서 다음 i 값을 준비함
    - 제5회: i 값 5를 4로 나누면 나머지는 1(거짓) → hap += 5 수행
    - 제6회: ……

LAB 주사위 3개를 동시에 던져 동일한 숫자 나오기

주사위 3개를 동시에 던져 모두 동일한 숫자가 나와야만 반복문을 탈출할 수 있는 게임을 만들어 봅시다. 몇 번을 던져야 3개의 주사위에서 동일한 숫자가 나올까요?

몇 번을 던져야 3개의 주사위가 동시에 같은 수가 나오는지 알 수가 없으므로, 일단은 무한 루프로 주사위를 던지겠습니다. 그리고 무한 루프 안에서 조건문으로 3개의 주사위가 같은 숫자면 무한 루프를 빠져나가도록 하겠습니다

실행 결과

```
3개 주사위는 모두 4 입니다.
같은 숫자가 나오기까지 132번 던졌습니다.
```

LAB 주사위 3개를 동시에 던져 동일한 숫자 나오기

1. lab06-03.py 파일을 만들고, 무작위로 주사위를 던지기 위해 random을 임포트한 후, 주사위를 던진 횟수를 저장할 변수와 주사위 3개도 선언하기

```
import random

count = 0
dice1, dice2, dice3 = 0, 0, 0
```

2. 3개의 주사위 던지기를 무한 반복하면서 주사위 던진 횟수를 1씩 증가시키기

```
while True :
    count += 1
    dice1 = random.randint(1,6)
    dice2 = random.randint(1,6)
    dice3 = random.randint(1,6)
```

LAB 주사위 3개를 동시에 던져 동일한 숫자 나오기

3. 3개의 주사위 숫자가 같다면 break문으로 무한 반복을 탈출하기

```
    if (dice1 == dice2) and (dice2 == dice3) :
        break
```

4. 주사위 3개에서 동시에 나온 숫자와 횟수를 출력하기

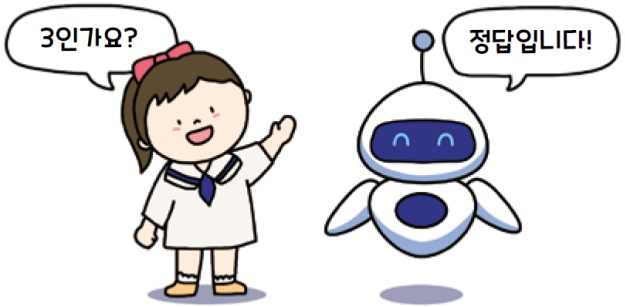
```
print("3개 주사위는 모두", dice1, "입니다.")
print("같은 숫자가 나오기까지", count, "번 던졌습니다.")
```

5. <Ctrl>+<S>를 눌러 저장한 후, <F5>를 눌러 실행 결과 확인하기

LAB 컴퓨터와 인간의 숫자 맞히기 대결

컴퓨터가 1~5까지 숫자 중 한 가지를 생각하면, 사람이 그 숫자를 맞히는 게임입니다. 단 10번 안에는 맞혀야 합니다.

이세돌 9단과 알파고의 바둑 대결만큼은 아니지만, 컴퓨터와 인간의 대결이라고 볼 수 있습니다. 숫자를 맞힌다면 break문으로 반복문을 빠져 나가고, 맞히지 못하면 continue문으로 다시 반복문의 처음 부분으로 돌아가도록 구현합니다



실행 결과

```
게임 1 회: 컴퓨터가 생각한 숫자는 ? 3 ●——— 사용자 입력
 아까워요. 4 였는데요.  다시 해보세요. ㅠ
게임 2 회: 컴퓨터가 생각한 숫자는 ? 2 ●——— 사용자 입력
 아까워요. 1 였는데요.  다시 해보세요. ㅠ
게임 3 회: 컴퓨터가 생각한 숫자는 ? 4 ●——— 사용자 입력
 맞혔네요. 축하합니다 !!
게임을 마칩니다.
```

LAB 컴퓨터와 인간의 숫자 맞히기 대결

1. lab06-04.py 파일을 만들고, 컴퓨터가 무작위로 숫자를 고르도록 random을 임포트한 후, 컴퓨터가 뽑은 숫자와 인간이 고른 숫자를 저장하기 위한 변수를 선언하기

```
import random

computer, user = 0, 0
```

2. 게임 전체 횟수인 10회를 반복하기. 컴퓨터가 1~5 중 랜덤한 숫자를 고르고, 게임의 현재 횟수를 출력하면서 사용자에게 숫자를 입력받기

```
for i in range(1, 11, 1) :
    computer = random.randint(1, 5)
    print ("게임", i, "회:", end='')
    user = int(input("컴퓨터가 생각한 숫자는 ? "))
```

LAB 컴퓨터와 인간의 숫자 맞히기 대결

3. 컴퓨터가 고른 숫자와 사용자가 생각한 숫자가 같다면 정답 메시지를 출력하고, for문을 빠져나가기
   - 숫자가 틀리면 오답 메시지를 출력하고 다시 반복문의 처음으로 올라감

```
    if computer == user :
        print(" 맞혔네요. 축하합니다 !! ")
        break
    else :
        print(" 아까워요. ", computer, "였는데요. 다시 해보세요. ㅠ")
        continue
```

4. 정답을 맞히거나, 10번의 기회가 끝나면 게임을 마친다는 메시지를 출력합니다.

```
print("게임을 마칩니다.")
```

5. <Ctrl>+<S>를 눌러 저장한 후, <F5>를 눌러 실행 결과 확인하기

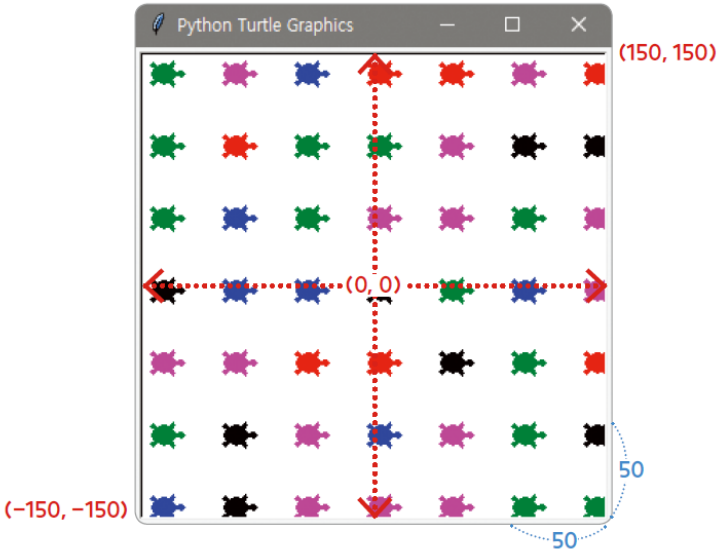
[실전 예제]
거북이 무늬의 벽지 만들기

실전 예제 거북이 문의의 벽지 만들기

[문제]

흰 벽지에 알록달록한 거북이 모양의 무늬를 가지런하게 그려보자.

- 벽지의 크기는 가로 300, 세로 300으로 지정
- 거북이는 약 50 거리만큼 떨어져서 다양한 색상의 거북이를 그림
- 중첩 for문을 활용해서 코드를 작성



실전 예제 거북이 문의의 벽지 만들기

[해결]

```
import turtle
import random

turtle.shape("turtle")
colors = ['red','green','magenta', 'blue', 'black']
turtle.penup( )
turtle.screensize(300,300)
turtle.setup(330,330)

for i in range(7) :
    for k in range(7) :
        x = i*50 – 150
        y = k*50 - 150
        turtle.goto( x, y )
        turtle.color(random.choice(colors))
        turtle.stamp( )

 turtle.done( )
```

Thank you!

한빛아카데미 Hanbit Academy, Inc.

Copyright© 2021 Hanbit Academy, Inc.
All rights reserved.