


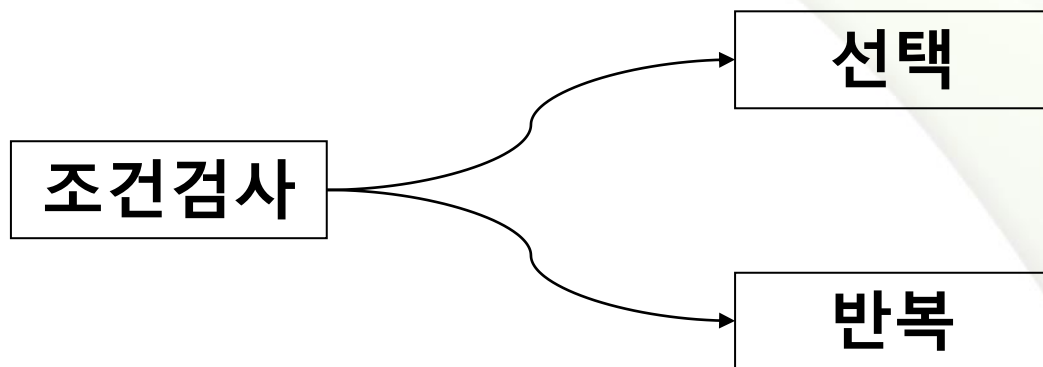
# 제어문

The background features a large, flowing green wave that starts from the left, peaks in the upper middle, and then descends towards the bottom right. The wave has a gradient, with lighter green at the top and darker green at the bottom. A solid dark green horizontal bar runs across the very bottom of the image.

# 제어문

- 프로그램의 실행 흐름을 논리적으로 제어하는 구문

- 동작 구조



- 지원 문장

- » 선택문 : if문

- » 반복문 : while문, for문

# 선택문 (if)

## ▪ 단순 if문

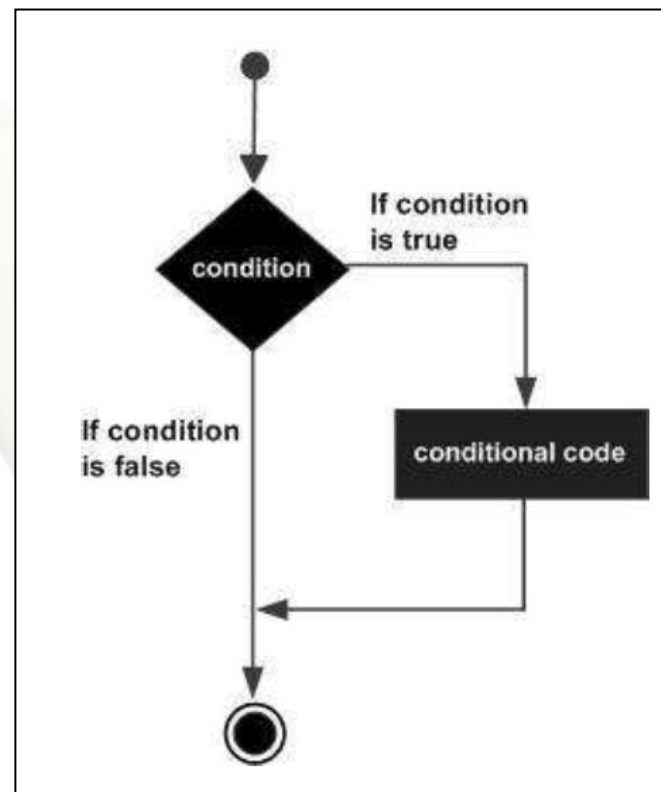
» 조건 검사 결과가 참일때 제어 대상 실행문을 실행하는 선택문

```
if expression:  
    statement(s)
```

- › expression이 참이면 문장 실행
- › expression이 거짓이면 문장 실행 생략

## » 예제

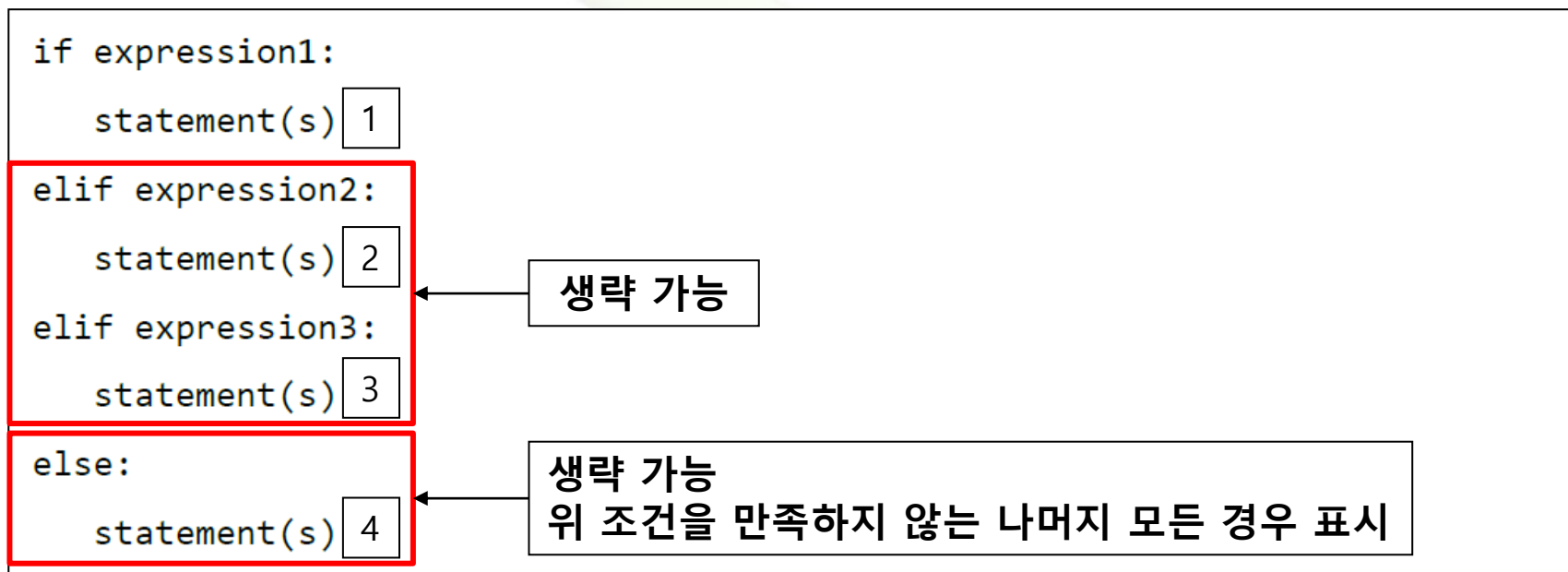
```
var1 = 100  
if var1:  
    print ("1 - Got a true expression value")  
    print (var1)  
var2 = 0  
if var2:  
    print ("2 - Got a true expression value")  
    print (var2)  
    print ("Good bye!")
```



# 선택문 (if)

## ■ 다중 선택 if문

» 여러 실행문 중 조건이 참이 되는 하나의 문장을 실행하는 선택문



- › expression1이 참일 경우 문장 1 실행
- › expression2가 참일 경우 문장 2 실행
- › expression3이 참일 경우 문장 3 실행
- › 나머지 모든 경우 문장 4 실행

# 선택문 (if)

## ■ 다중 선택 if문 예제

```
amount=int(input("Enter amount: "))

if amount<1000:
    discount=amount*0.05
    print ("Discount",discount)
elif amount<5000:
    discount=amount*0.10
    print ("Discount",discount)
else:
    discount=amount*0.15
    print ("Discount",discount)
print ("Net payable:",amount-discount)
```



```
Enter amount: 600
Discount 30.0
Net payable: 570.0

Enter amount: 3000
Discount 300.0
Net payable: 2700.0

Enter amount: 6000
Discount 900.0
Net payable: 5100.0|
```

# 조건식 만들기

- 자료형에 따라 참/거짓을 판단하는 조건식

자료형	참	거짓
숫자	0이 아닌 숫자	0
문자열	"abc"	""
리스트	[1,2,3]	[]
튜플	(1,2,3)	()
딕셔너리	{"a":"b"}	{}

```
>>> money = 1
>>> if money:
...     print("택시를 타고 가라")
... else:
...     print("걸어 가라")
...
택시를 타고 가라
```

# 조건식 만들기

- 관계 연산자 (비교 연산자)를 사용하는 조건식
  - » 두 데이터의 대소를 비교해서 참/거짓 값을 반환하는 연산자
  - » 조건식에 많이 사용되는 연산자

비교연산자	설명
$x < y$	x가 y보다 작으면 참 크거나 같으면 거짓
$x > y$	x가 y보다 크면 참 작거나 같으면 거짓
$x == y$	x와 y가 같으면 참 다르면 거짓
$x != y$	x와 y가 같지 않으면 참 같으면 거짓
$x >= y$	x가 y보다 크거나 같으면 참 작으면 거짓
$x <= y$	x가 y보다 작거나 같으면 참 크면 거짓

```
>>> money = 2000
>>> if money >= 3000:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
걸어가라
>>>
```

# 선택문 (if)

- and, or, not 을 사용하는 조건식
  - » 두 논리값을 결합해서 하나의 논리 값을 반환 (and, or)
  - » 대상 논리값의 반대 논리값 반환 (not)

연산자	설명
x or y	x와 y 둘중에 하나만 참이면 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

```
>>> money = 2000
>>> card = 1
>>> if money >= 3000 or card:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```



# 선택문 (if)

- 데이터의 포함 여부로 조건 분기

- » `x in s` / `x not in s`

in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

```
>>> 1 in [1, 2, 3]
True
>>> 1 not in [1, 2, 3]
False
>>> pocket = ['paper', 'cellphone', 'money']
>>> if 'money' in pocket:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```

# 선택문 (if)

- 조건 분기된 실행문이 없는 경우 pass 구문 사용

```
>>> pocket = ['paper', 'money', 'cellphone']  
>>> if 'money' in pocket:  
...     pass  
... else:  
...     print("카드를 꺼내라")  
...
```

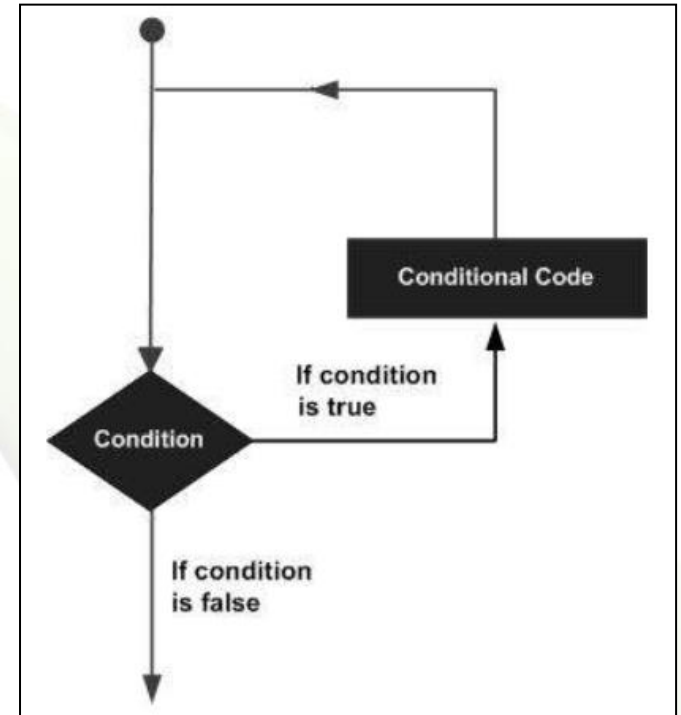
# 반복문 (while)

- 조건에 따라 대상 실행문 집합을 여러 번 반복할 수 있는 구문
- 형식

```
while <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    <수행할 문장3>  
    ...
```

- 예제

```
>>> treeHit = 0  
>>> while treeHit < 10:  
...     treeHit = treeHit +1  
...     print("나무를 %d번 찍었습니다." % treeHit)  
...     if treeHit == 10:  
...         print("나무 넘어갑니다.")  
...  
나무를 1번 찍었습니다.  
나무를 2번 찍었습니다.  
나무를 3번 찍었습니다.  
...  
나무를 10번 찍었습니다.  
나무 넘어갑니다.
```



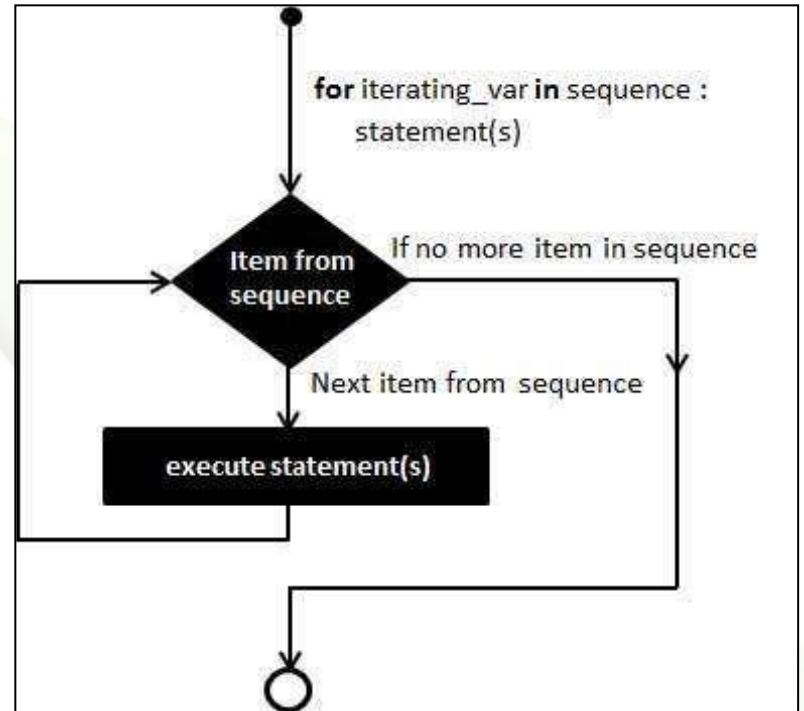
# 반복문 (for)

- 반복의 사이클이 명확한 경우에 적합한 반복 구문

```
for 변수 in 리스트(또는 튜플, 문자열):  
    수행할 문장1  
    수행할 문장2  
    ...
```

```
>>> test_list = ['one', 'two', 'three']  
>>> for i in test_list:  
...     print(i)  
...  
one  
two  
three
```

```
>>> a = [(1,2), (3,4), (5,6)]  
>>> for (first, last) in a:  
...     print(first + last)  
...  
3  
7  
11
```



# range 함수

- range 함수 → 범위를 만드는 함수로 for문과 함께 사용

```
>>> a = range(10)
>>> a
range(0, 10)
>>> a = range(1, 11)
>>> a
range(1, 11)
```

```
>>> sum = 0
>>> for i in range(1, 11):
...     sum = sum + i
...
>>> print(sum)
55
```

# iterator

- 리스트, 튜플 등의 컬렉션 객체에 포함된 모든 요소들을 순차적으로 접근할 수 있는 객체
- 파이썬의 iterator는 `iter()`, `next()` 두 개의 함수 제공
  - » `iter()` 함수는 반복자를 만드는 함수
  - » `next()` 함수는 반복자의 다음 요소를 반환하는 함수 (없으면 `None`)

```
import sys

mylist=[1,2,3,4]
it = iter(mylist)
print (next(it))

it = iter(mylist)
for x in it:
    print (x, end=" ")
```

```
it = iter(mylist)
print()
while True:
    try:
        print (next(it))
    except StopIteration:
        break;
```

# list 내포

- list 내포 → list와 for문을 함께 사용

» 형식

```
[표현식 for 항목 in 반복가능객체 if 조건]
```

```
[표현식 for 항목1 in 반복가능객체1 if 조건1  
      for 항목2 in 반복가능객체2 if 조건2  
      ...  
      for 항목n in 반복가능객체n if 조건n]
```

» 예제

```
>>> result = [num * 3 for num in a]  
>>> print(result)  
[3, 6, 9, 12]
```

```
>>> result = [num * 3 for num in a if num % 2 == 0]  
>>> print(result)  
[6, 12]
```

# else 구문을 사용하는 반복문

- 반복문에 else를 사용하면 반복문이 정상 종료된 후 else의 실행문이 실행됨
- 예제

```
count = 0
while count < 5:
    print (count, " is less than 5")
    count = count + 1
else:
    print (count, " is not less than 5")
```

```
0 is less than 5
1 is less than 5
2 is less than 5
3 is less than 5
4 is less than 5
5 is not less than 5
```



# break 문

- 반복문 탈출 (강제 종료) → break 문 사용

```
coffee = 10
while True:
    money = int(input("돈을 넣어 주세요: "))
    if money == 300:
        print("커피를 줍니다.")
        coffee = coffee - 1
    elif money > 300:
        print("거스름돈 %d를 주고 커피를 줍니다." % (money - 300))
        coffee = coffee - 1
    else:
        print("돈을 다시 돌려주고 커피를 주지 않습니다.")
        print("남은 커피의 양은 %d개 입니다." % coffee)

    if not coffee:
        print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")
        break
```

# continue 문

- 다음 반복으로 점프 (반복문의 처음으로 실행 위치 이동) → continue

```
>>> a = 0
>>> while a < 10:
...     a = a+1
...     if a % 2 == 0: continue
...     print(a)
...
1
3
5
7
9
```

# pass 문

- 구문 규칙상 문장이 필요하지만 실제로는 작성할 코드가 없는 경우 사용
  - » pass문이 있는 곳에서는 어떤 실행도 발생하지 않음
  - » 나중에 작성될 코드에 대한 Placeholder 용도로 사용

```
for letter in 'Python':  
    if letter == 'h':  
        pass  
        print ('This is pass block')  
    print ('Current Letter :', letter)  
  
print ("Good bye!")
```



```
Current Letter : P  
Current Letter : y  
Current Letter : t  
This is pass block  
Current Letter : h  
Current Letter : o  
Current Letter : n  
Good bye!
```

# 반복문 관련 함수

## ▪ filter

» 원본 리스트의 요소 중 조건에 맞는 요소를 추출해서 반복 객체 반환

```
>>> for n in filter(lambda x: x > 20, [15, 25, 35, 45]):  
>>>     print(n, end=' ')...  
25 35 45
```

```
>>> list(filter(lambda x: x > 20, [15, 25, 35, 45]))  
25 35 45
```

## ▪ zip

» 두 개 이상의 리스트의 요소를 결합한 요소를 추출해서 반복 객체 반환

```
>>> for x, y in zip([10, 20, 30], [40, 50, 60]):  
>>>     print((x, y), end=' ')  
(10, 40) (20, 50) (30, 60)
```

# 반복문 관련 함수

- map

- » 원본 리스트의 요소를 변환해서 반복 객체 반환

```
>>> for x in map(lambda x: x**2, [1, 2, 3, 4, 5]):  
>>>     print(x, end=' ')  
1 4 9 16 25
```