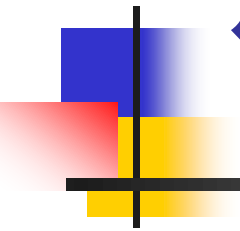
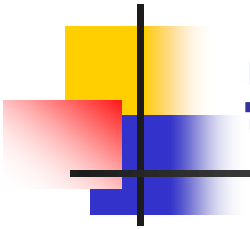


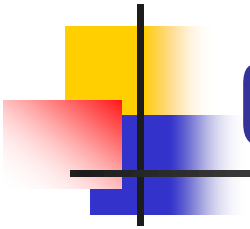
03장 프로그램의 구조를 쌓는다! 제어문

자료형을 바탕으로 제어문을 이용하여 프로그램의 구조를 만들어 보자



목차

- If 문
- while 문
- for 문
  - 리스트 내포(List comprehension)

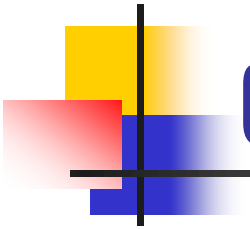


03-1 If 문

조건을 판단하여 해당 조건에 맞는 상황을 수행하는 데 쓰는 것이 if문

돈이 있으면 택시를 타고, 돈이 없으면 걸어 간다

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



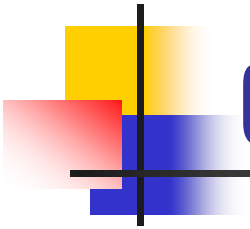
03-1 If 문

if문의 기본구조

```
if 조건문:
    수행할 문장1
    수행할 문장2
    ...
else:
    수행할 문장A
    수행할 문장B
    ...
```

조건문을 테스트해서 참이면 if문 바로 다음 문장(if 블록)들을 수행하고,
조건문이 거짓이면 else문 다음 문장(else 블록)들을 수행하게 된다.

else문은 if문 없이 독립적으로 사용할 수 없다.



03-1 If 문

들여쓰기

if 조건문: 바로 아래 문장부터 if문에 속하는 모든 문장에 들여쓰기(indentation)를 해주어야 한다.

```
if 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
```

03-1 If 문

들여쓰기 오류

```
if 조건문:
    수행할 문장1
수행할 문장2
    수행할 문장3
```

```
>>> if money:
...     print("택시를")
... print("타고")
File "<stdin>", line 4
print("가자")
^
SyntaxError: invalid syntax
```

"수행할 문장3"을 들여쓰기 했지만 "수행할 문장1"이나 "수행할 문장2"와 들여쓰기의 너비가 다르다. 즉 들여쓰기는 언제나 같은 너비로 해야 한다.

```
if 조건문:
    수행할 문장1
    수행할 문장2
       수행할 문장3

money = True
if money:
    print("택시를")
    print("타고")
       print("가라")
```

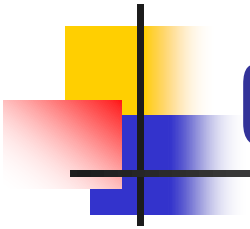
03-1 If 문

조건문

```
>>> money = True
>>> if money:
```

if 조건문에서 "조건문"이란 참과 거짓을 판단하는 문장을 말한다.

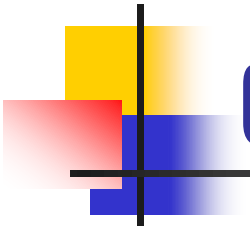
money는 True이기 때문에 조건이 참이 되어 if문 다음 문장을 수행한다.



03-1 If 문

비교 연산자

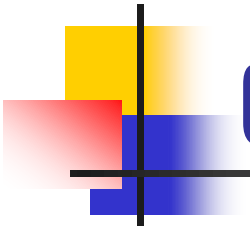
비교연산자	설명
x < y	x가 y보다 작다
x > y	x가 y보다 크다
x == y	x와 y가 같다
x != y	x와 y가 같지 않다
x >= y	x가 y보다 크거나 같다
x <= y	x가 y보다 작거나 같다



03-1 If 문

비교 연산자

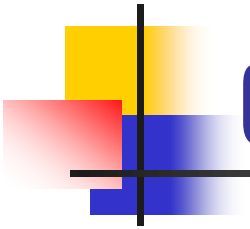
```
>>> x = 3
>>> y = 2
>>> x > y
True
>>> x < y
False
>>> x == y
False
>>> x != y
True
```



03-1 If 문

만약 3000원 이상의 돈을 가지고 있으면 택시를 타고 그렇지 않으면 걸어 가라

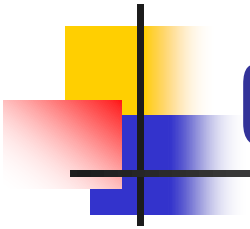
```
>>> money = 2000
>>> if money >= 3000:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
걸어가라
```



03-1 If 문

and, or, 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



03-1 If 문

돈이 3000원 이상 있거나 카드가 있다면 택시를 타고 그렇지 않으면 걸어 가라

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

03-1 If 문

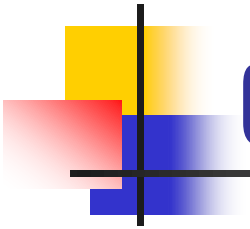
in	not in
x in 리스트	x not in 리스트
x in 튜플	x not in 튜플
x in 문자열	x not in 문자열

x in s, x not in s

```
>>> 1 in [1, 2, 3]
True
>>> 1 not in [1, 2, 3]
False

>>> 'a' in ('a', 'b', 'c')
True
>>> 'j' not in 'python'
True
```

[1, 2, 3]이라는 리스트 안에 1이 있는가?" 조건문이다.
1은 [1, 2, 3] 안에 있으므로 참이 되어 True를 돌려준다.



03-1 If 문

만약 주머니에 돈이 있으면 택시를 타고, 없으면 걸어 가라

```
>>> pocket = ['paper', 'cellphone', 'money']
>>> if 'money' in pocket:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```

03-1 If 문

조건문에서 아무 일도 하지 않게 설정하고 싶다면?

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

"주머니에 돈이 있으면 가만히 있고 주머니에 돈이 없으면 카드를 꺼내라."

조건문의 참, 거짓에 따라 실행할 행동을 정의할 때, 아무런 일도 하지 않도록 설정하고 싶을 때가 있다.

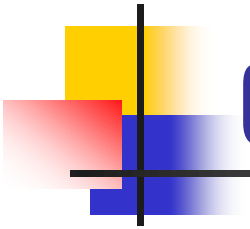
이럴 때 사용하는 것이 **pass**

03-1 If 문

다중 조건 판단 elif

"주머니에 돈이 있으면 택시를 타고, 주머니에 돈은 없지만 카드가 있으면 택시를 타고, 돈도 없고 카드도 없으면 걸어 가라."

```
>>> pocket = ['paper', 'cellphone']
>>> card = 1
>>> if 'money' in pocket:
...     print("택시를 타고가라")
... elif card:
...     print("택시를 타고가라")
... else:
...     print("걸어가라")
...
택시를 타고가라
```



03-1 If 문

```
If <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
else:
   <수행할 문장1>
   <수행할 문장2>
   ...
```

03-1 If 문

조건부 표현식

조건문이 **참**인 경우 **if** 조건문 **else** 조건문이 **거짓**인 경우

```
if score >= 60:
    message = "success"
else:
    message = "failure"
```

파이썬의 조건부 표현식(conditional expression)을 사용하면 위 코드를 다음과 같이 간단히 표현할 수 있다.

```
message = "success" if score >= 60 else "failure"
```

03-2 while 문

while문의 기본 구조

반복해서 문장을 수행해야 할 경우 while문을 사용

조건문이 참인 동안에 while문 아래의 문장이 반복해서 수행된다.

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

03-2 while 문

열 번 찍어 안 넘어 가는 나무 없다

treeHit가 10보다 작은 동안에 while문 안의 문장을 계속 수행한다

```
>>> 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.
…
나무 넘어갑니다.
```

03-2 while 문

여러 가지 선택지 중 하나를 선택해서 입력 받는 예제

```
>>> prompt = """
... 1. Add
... 2. Del
... 3. List
... 4. Quit
...
... Enter number: """
>>> number = 0
>>> while number != 4:
...     print(prompt)
...     number = int(input())
...
1. Add
2. Del
3. List
4. Quit

Enter number:
```

여러 줄짜리 문자열을 입력

while문을 보면 number가 4가 아닌 동안 prompt를 출력하고 사용자로부터 번호를 입력 받는다.
사용자가 값 4를 입력하지 않으면 계속해서 prompt를 출력한다.

03-2 while 문

break

while문 강제로 빠져나가기

```
>>> coffee = 10
>>> money = 300
>>> while money:
...     print("돈을 받았으니 커피를 줍니다.")
...     coffee = coffee -1
...     print("남은 커피의 양은 %d개입니다." % coffee)
...     if not coffee:
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...         break
...
```

03-2 while 문

```
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 coffee == 0:
        print("커피가 다 떨어졌습니다. 판매를 중지 합니다.")
        break
```

03-2 while 문

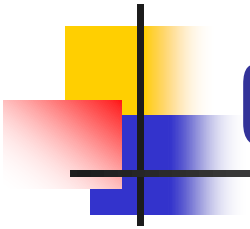
continue

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

while문을 빠져나가지 않고 while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우 continue문 사용

1부터 10까지의 숫자 중에서 홀수만 출력하는 것을 while문을 사용해서 작성

continue문은 while문의 맨 처음(조건문: a<10)으로 돌아가게 하는 명령어

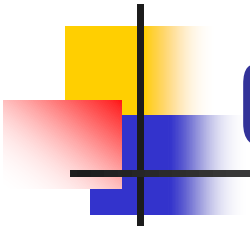


03-2 while 문

무한 루프

무한 루프란 무한히 반복한다는 의미

```
while True:
    수행할 문장1
    수행할 문장2
    ...
```

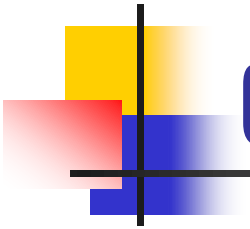


03-3 for 문

for문의 기본 구조

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 "수행할 문장1", "수행할 문장2" 등이 수행된다.

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



03-3 for 문

전형적인 for문

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

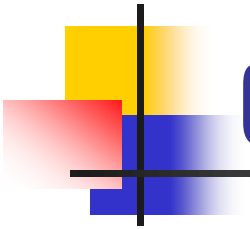
03-3 for 문

다양한 for문의 사용

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

a 리스트의 요솟값이 튜플이기 때문에 각각의 요소가 자동으로 (first, last) 변수에 대입된다.

튜플을 사용한 변수값 대입 방법과 매우 비슷한 경우
>>> (first, last) = (1, 2)



03-3 for 문

60점이 넘으면 합격이고 그렇지 않으면 불합격

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)
```

03-3 for 문

for문과 continue

60점 이상인 사람에게는 축하 메시지를 보내고 나머지 사람에게는 아무 메시지도 전하지 않는 프로그램

```
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number +1
    if mark < 60: continue
    print("%d번 학생 축하합니다. 합격입니다. " % number)
```

03-3 for 문

for와 함께 자주 사용하는 range함수

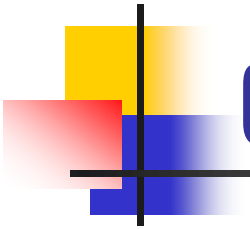
```
>>> a = range(10)
>>> a
range(0, 10)

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

for문은 **숫자 리스트를 자동으로 만들어 주는 range 함수**와 함께 사용하는 경우가 많다.

range(시작 숫자, 끝 숫자) 형태를 사용하는데, 이때 끝 숫자는 포함되지 않는다.
range(10)은 0부터 10 미만의 숫자를 포함하는 range 객체를 만들어 준다.

1부터 10까지 더하기
range(1, 11)은 숫자 1부터 10까지(1 이상 11 미만)의 숫자를 데이터로 갖는 객체



03-3 for 문

"60점 이상이면 합격"이라는 문장을 출력하는 예제 - range 함수 사용

```
marks = [90, 25, 67, 45, 80]
for number in range(len(marks)):
    if marks[number] < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다." % (number+1))
```

len(marks)는 5
range(len(marks))는 range(5)
number 변수에는 차례로 0부터 4까지의 숫자가 대입

03-3 for 문

구구단

```
>>> for i in range(2,10):
...     for j in range(1, 10):
...         print(i*j, end=" ")
...     print('')
...
2 4 6 8 10 12 14 16 18
3 6 9 12 15 18 21 24 27
4 8 12 16 20 24 28 32 36
5 10 15 20 25 30 35 40 45
6 12 18 24 30 36 42 48 54
7 14 21 28 35 42 49 56 63
8 16 24 32 40 48 56 64 72
9 18 27 36 45 54 63 72 81
```

매개변수 end를 넣어 준 이유는 해당 결괏값을 출력할 때 다음줄로 넘기지 않고 그 줄에 계속해서 출력하기 위해서.

print(' ')는 2단, 3단 등을 구분하기 위해 두 번째 for문이 끝나면 결괏값을 다음 줄부터 출력하게 해주는 문장

리스트 안에 for문을 포함하는 리스트 내포(List comprehension) 사용

[표현식 for 항목 in 반복가능객체 if 조건문]

03-3 for 문

리스트 내포(List comprehension)

```
>>> a = [1,2,3,4]
>>> result = []
>>> for num in a:
...     result.append(num*3)
...
>>> print(result)
[3, 6, 9, 12]

>>> 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]
```

리스트 내포 안에 "if 조건"을 사용할 수 있다.

03-3 for 문

for문을 2개 이상 사용하는 것도 가능.

```
[표현식 for 항목1 in 반복가능객체1 if 조건문1
        for 항목2 in 반복가능객체2 if 조건문2
        ...
        for 항목n in 반복가능객체n if 조건문n]
```

구구단의 모든 결과를 리스트에 담고 싶다면 리스트 내포를 사용하여 구현

```
>>> result = [x*y for x in range(2,10)
...                for y in range(1,10)]
>>> print(result)
[2, 4, 6, 8, 10, 12, 14, 16, 18, 3, 6, 9, 12, 15, 18, 21, 24, 27,
4, 8, 12, 16,
20, 24, 28, 32, 36, 5, 10, 15, 20, 25, 30, 35, 40, 45, 6, 12, 18,
24, 30, 36, 42
, 48, 54, 7, 14, 21, 28, 35, 42, 49, 56, 63, 8, 16, 24, 32, 40,
48, 56, 64, 72,
9, 18, 27, 36, 45, 54, 63, 72, 81]
```

Q1
다음 코드의 결괏값은 무엇일까?

```
a = "Life is too short, you need python"

if "wife" in a: print("wife")
elif "python" in a and "you" not in a: print("python")
elif "shirt" not in a: print("shirt")
elif "need" in a: print("need")
else: print("none")
```

Q2
while문을 사용해 1부터 1000까지의 자연수 중 3의 배수의 합을 구해 보자.

Q3
while문을 사용하여 다음과 같이 별(*)을 표시하는 프로그램을 작성해 보자.

```
*
**
***
****
*****
```

Q4
for문을 사용해 1부터 100까지의 숫자를 출력해 보자.

Q5
A 학급에 총 10명의 학생이 있다. 이 학생들의 중간고사 점수는 다음과 같다.

[70, 60, 55, 75, 95, 90, 80, 80, 85, 100]

for문을 사용하여 A 학급의 평균 점수를 구해 보자.

Q6
리스트 중에서 홀수에만 2를 곱하여 저장하는 다음 코드가 있다.

```
numbers = [1, 2, 3, 4, 5]
result = []
for n in numbers:
    if n % 2 == 1:
        result.append(n*2)
```

위 코드를 리스트 내포(list comprehension)를 사용하여 표현해 보자.