# 3장. 프로그램 구조, 제어문

# Contents

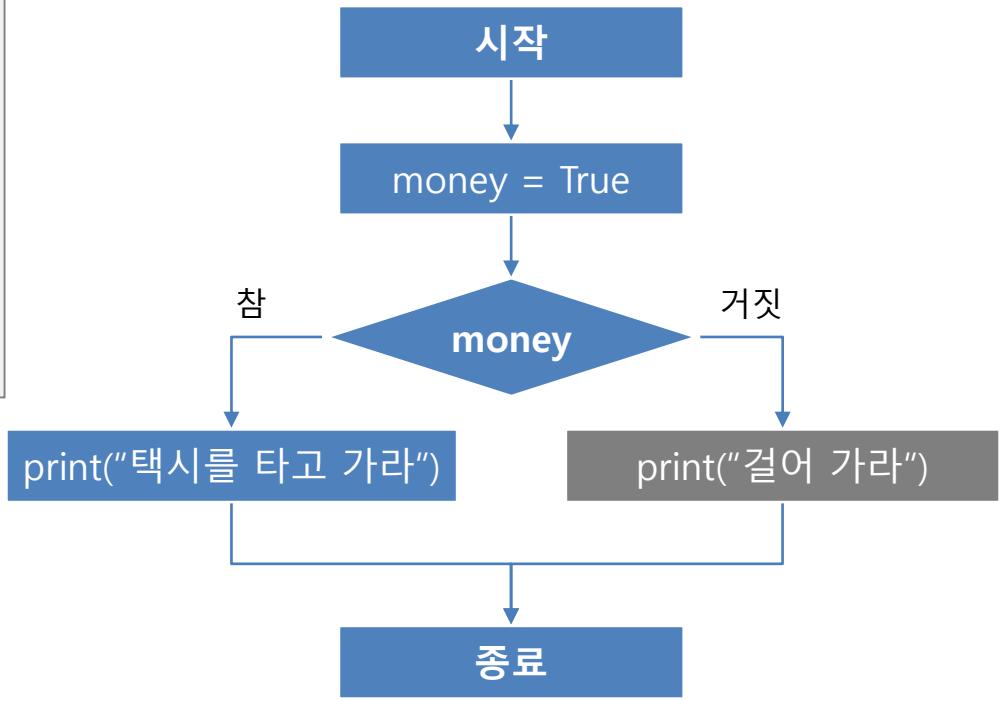
# I. if문

## ❖ if문의 왜 필요할까?

"돈이 있으면 택시를 타고, 돈이 없으면 걸어 간다."

✓ 주어진 조건을 판단하여 해당 조건에 맞는 상황을 수행하는데 if문이 쓰임

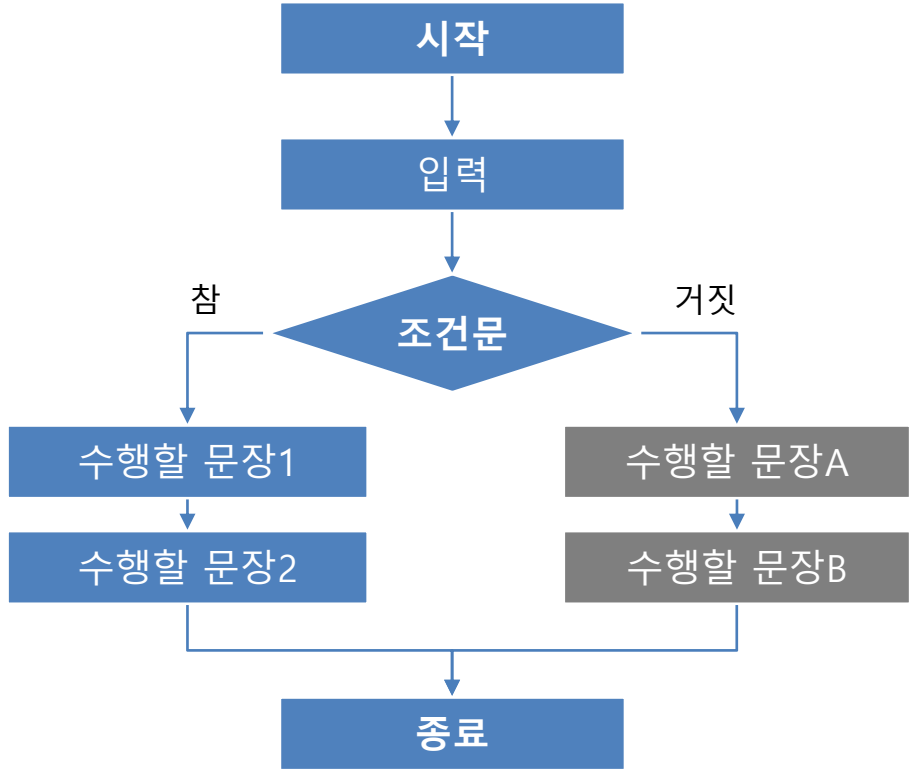
```
>>> money = 1
>>> if money:
...         print("택시를 타고 가라")
...     else:
...         print("걸어 가라")
...
택시를 타고 가라
```



# I. if문

## ❖ if문의 기본 구조

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




- ✓ if 조건문 뒤에는 반드시 콜론(:)이 붙임
- ✓ 조건문을 테스트해서 참이면 if문 바로 다음의 문장(if 블록)들을 수행하고, 조건문이 거짓이면 else문 다음 문장(else 블록)들을 수행하게 됨
- ✓ 블록은 들여쓰기로 해결함 (다른 언어는 { }로 기호 사용)
- ✓ else문은 if문 없이 독립적으로 사용할 수 없음

# I. if문

- **들여쓰기**

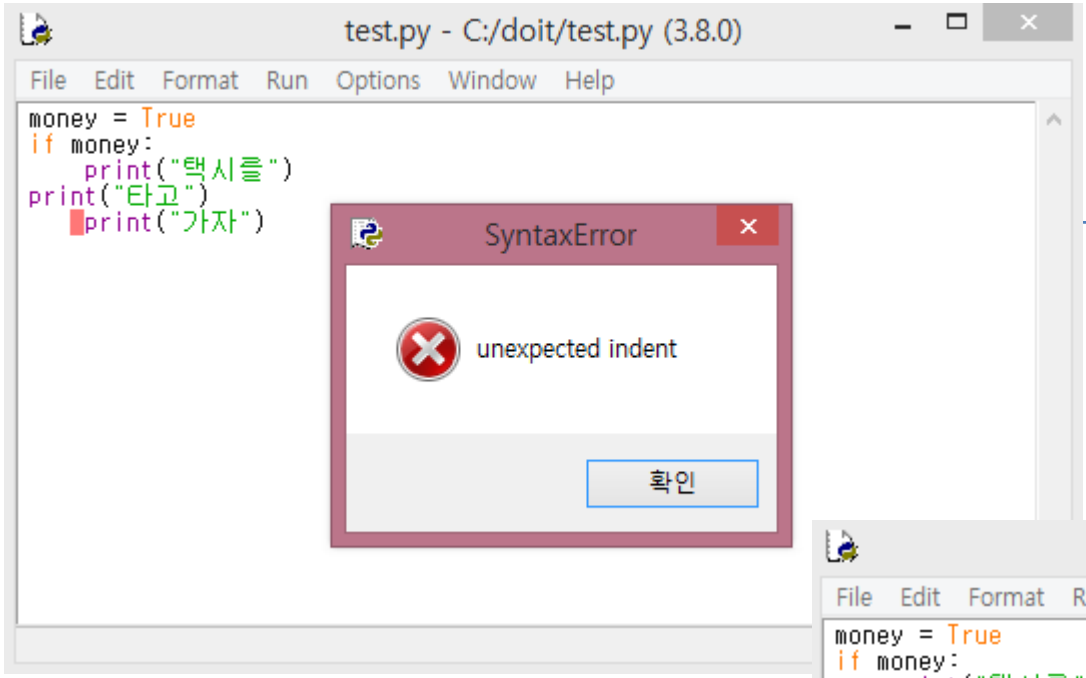
```
if 조건문:
    수행할 문장1
    수행할 문장2
    수행할 문장3
```

  - ✓ if문을 만들 때는 if 조건문: 바로 아래 문장부터 if 문에 속하는 모든 문장에 들여쓰기(indentation)를 해주어야 함
  - ✓ 들여쓰기는 공백(Spacebar) 4개나 탭(Tab)을 사용하며, 2가지를 혼용하지 말 것

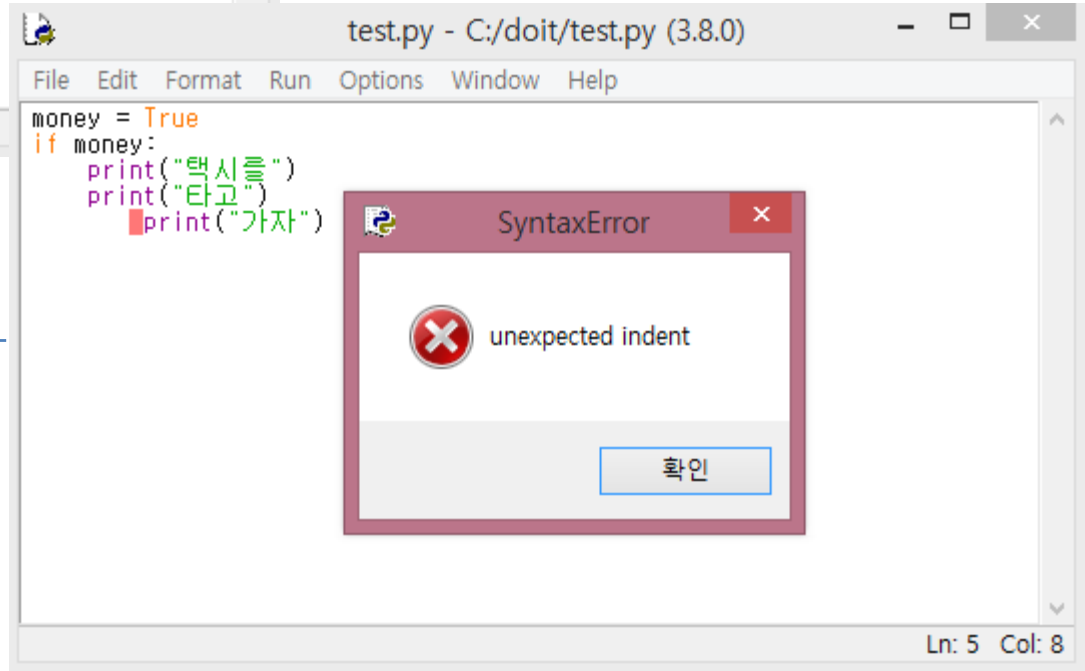
- **들여쓰기 오류**

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

  - ✓ 수행할 문장2와 같이 들여쓰기를 하지 않으면 오류가 발생함




---들여쓰기를 하지 않았기 때문에 오류가 발생함




들여쓰기 너비가 다르기 때문에 오류가 발생함--

# I. if문

## ❖ 조건문이란 무엇인가?

✓ if 조건문에서 '조건문'이란 참과 거짓을 판단하는 문장을 말함

```
>>> money = 1
>>> if money:
```

- 위의 예제에서 조건문은 money가 됨
- money는 1이기 때문에 참이 되어 if문 다음의 문장을 수행함

- **비교연산자**

조건문에 비교연산자(<, >, ==, !=, >=, <=)를 쓰는 경우가 많음

| 비교 연산자 | 설명 |
|---|---|
| x < y | x가 y보다 작음 |
| x > y | x가 y보다 큼 |
| x == y | x와 y가 같음 |
| x != y | x와 y가 같지 않음 |
| x >= y | x가 y보다 크거나 같음 |
| x <= y | x가 y보다 작거나 같음 |

```
>>> x = 3
>>> y = 2
>>> x > y
True
```

✓ x에 3을, y에 2를 대입한 다음에 x > y라는 조건문을 수행하면, x > y 조건문이 참이기 때문임 True를 돌려줌

```
>>> x < y
False
```

✓ 위의 조건은 거짓이기 때문에 False를 돌려줌

```
>>> x == y
False
```

✓ x와 y는 같지 않아 위의 조건문은 거짓임

```
>>> x != y
True
```

✓ x와 y는 같지 않으므로 위의 조건문은 참임

# I. if문

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

✓ 위와 같이 택시 예제를 바꾸면, 다음처럼 프로그래밍할 수 있음

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

- money >= 3000 조건문이 거짓이 되기 때문에 else문 다음 문장을 수행하게 됨

- **and, or, not**

조건을 판단하기 위해 사용하는 연산자로는 and, or, not이 있음

| 연산자 | 설명 |
|---|---|
| x or y | x와 y 둘 중에 하나만 참이면 참임 |
| x and y | x와 y 모두 참이어야 참임 |
| not x | x가 거짓이면 참임 |

✓ or 연산자의 사용법을 알아봄

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

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

- money가 2000이지만 card가 1이기 때문에 money >= 3000 or card 조건문이 참이 되어 if 다음의 '택시를 타고 가라' 문장이 출력됨

- **x in s, x not in s**

  다른 프로그래밍 언어에서 볼 수 없는 조건문들을 제공함

| in | not in |
|---|---|
| x in 리스트 | x not in 리스트 |
| x in 튜플 | x not in 튜플 |
| x in 문자열 | x not in 문자열 |

✓ in의 뜻이 '~안에'라는 것을 생각하면 쉽게 이해될 것임

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

- 첫 번째 예의 1은 [1, 2, 3] 안에 있으므로 참이 되어 True를 리턴 함
- 두 번째 예의 1은 [1, 2, 3] 안에 있으므로 False를 리턴 함

✓ 튜플과 문자열에 적용한 예

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

✓ 택시 예제에 in을 적용

만약 주머니에 돈이 있으면 택시를 타고, 없으면 걸어가라.

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

- ['paper', 'cellphone', 'money']라는 리스트 안에 'money'가 있으므로 'money' in pocke은 참이 되어서 if문 다음 문장이 수행됨

✓ 조건문에서 아무 일도 하지 않게 설정하고 싶다면?

주머니에 돈이 있으면 가만히 있고 주머니에 돈이 없으면 카드를 꺼내라.

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

- pocket이라는 리스트 안에 money라는 문자열이 있기 때문에 if문 다음 문장인 pass가 수행되고 아무런 결과값도 보여 주지 않음

## ❖ 다양한 조건을 판단하는 elif

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

- if와 else만으로 표현

```
>>> pocket = ['paper', 'cellphone']      ---주머니 안에 종이, 휴대폰이 있음
>>> card = 1                             ---카드를 가지고 있음
>>> if 'money' in pocket:
...         print("택시를 타고 가라")
...     else:
...         if card:
...             print("택시를 타고 가라")
...         else:
...             print("걸어가라")
...
택시를 타고 가라
```

✓ 이해하기 어렵고 산만한 느낌이 든다.

- 복잡함을 해결하기 위해 다중 조건 판단을 가능하게 하는 elif 사용

```
>>> pocket = ['paper', 'cellphone']
>>> card = 1
>>> if 'money' in pocket:                    ---주머니에 돈이 있으면
...         print("택시를 타고 가라")
...     elif card:                            ---주머니에 돈이 없고 카드가 있으면
...         print("택시를 타고 가라")
...     else:                                 ---주머니에 돈이 없고 카드도 없으면
...         print("걸어 가라")
...
택시를 타고 가라
```

- ✓ elif는 이전 조건문이 거짓일 때 수행됨
- ✓ elif는 개수에 제한 없이 사용할 수 있음

# I. if문

- if, elif, else를 모두 사용할 때의 기본 구조

```
if 조건문1:
    수행할 문장1-1
    수행할 문장1-22
    ...
elif 조건문2:
    수행할 문장2-1
    수행할 문장2-2
    ...
...
elif 조건문N:
    수행할 문장N-1
    수행할 문장N-2
    ...
...
else:
   수행할 문장A
   수행할 문자B
   ...
```

## ❖ if문을 한 줄로 작성하기

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

✓ 앞의 pass를 사용한 예를 보면 if문 다음에 수행할 문장이 한 줄이고, else문 다음에 수행할 문장도 한 줄 밖에 되지 않음

✓ 수행할 문장이 한 줄일 때 조금 더 간략하게 코드를 작성하는 방법이 있음

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

✓ if문 다음 수행할 문장을 콜론(:) 뒤에 적어 주었고, else문도 마찬가지임

## ❖ 조건부 표현식

```
score = 70
if score >= 60:
    message = "success"
else:
    message = "failure"
```

✓ score가 60 이상일 경우 messag에 문자열 "success"를, 아닐 경우에는 "failure"를 대입하는 코드임

✓ 조건부 표현식(conditional expression)을 사용하면 간단히 표현할 수 있음

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

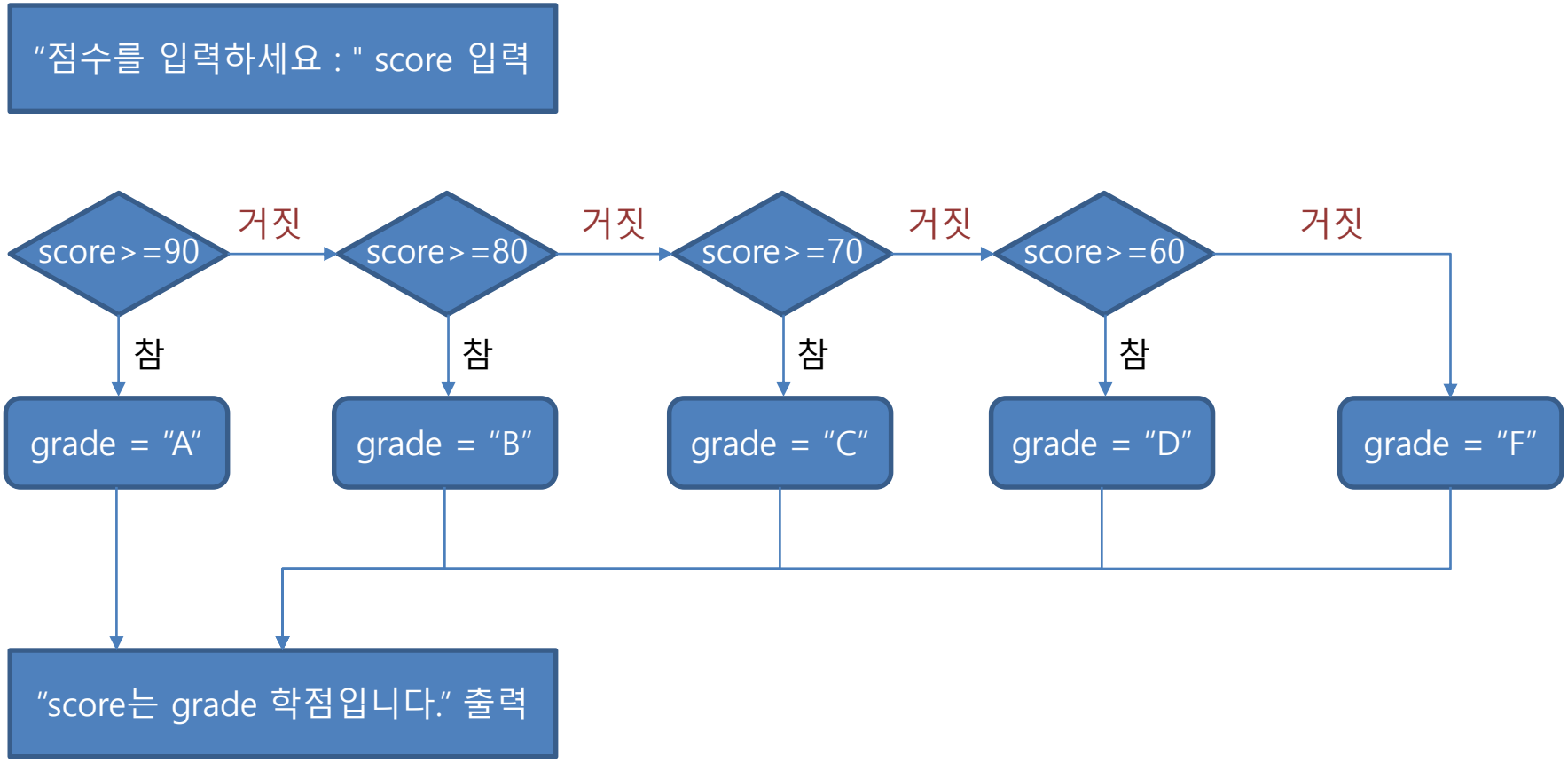
✓ 조건부 표현식 정의

조건문이 참인 경우 if 조건문 else 조건문이 거짓인 경우

- 가독성에 유리하고 한 줄로 작성할 수 있어 활용성이 좋음

## 응용예제01. 학점 세분화 프로그램

✓ 점수를 입력받은 후 90점 이상은 A, 80점 이상은 B, 70점 이상은 C, 60점 이상은 D, 나머지는 F로 처리하는 프로그램을 구현해 보자



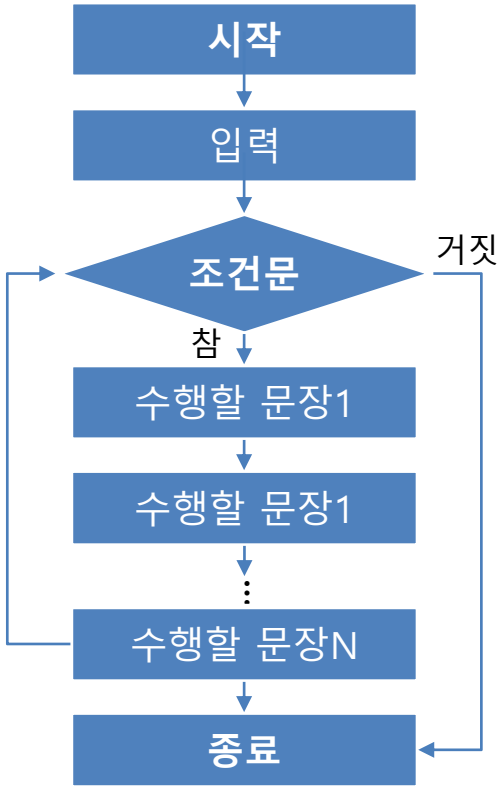
# II. while문

## ❖ while문의 기본 구조

✓ 반복해서 문장을 수행해야 할 경우 while문을 사용함

✓ while문의 기본구조

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




✓ while문은 조건문이 참인 동안에 while문 아래에 속하는 문장들이 반복해서 수행됨

# II. while문

✓ '열 번 찍어 안 넘어가는 나무 없다'는 속담을 while문으로 만든 예

```
>>> treeHit = 0                    ← 나무를 찍은 횟수
>>> while treeHit < 10:            ← 나무를 찍은 횟수가 10보다 작은 동안 반복함
...         treeHit = treeHit +1    ← 나무를 찍은 횟수 1씩 증가
...         print("나무를 %d번 찍었습니다." % treeHit)
...         if treeHit == 10:       ← 나무를 10번 찍으면
...             print("나무 넘어갑니다.")
...
나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

# II. while문

✓ while문이 반복되는 과정을 순서대로 정리한 표

| treeHit | 조건문 | 조건 판단 | 수행하는 문장 | while문 |
|---|---|---|---|---|
| 0 | 0 < 10 | 참 | treeHit = treeHit + 1<br>나무를 1번 찍었습니다 | 반복 |
| 1 | 1 < 10 | 참 | treeHit = treeHit + 1<br>나무를 2번 찍었습니다 | 반복 |
| 2 | 2 < 10 | 참 | treeHit = treeHit + 1<br>나무를 3번 찍었습니다 | 반복 |
| 3 | 3 < 10 | 참 | treeHit = treeHit + 1<br>나무를 4번 찍었습니다 | 반복 |
| … | … | … | … | … |
| 8 | 8 < 10 | 참 | treeHit = treeHit + 1<br>나무를 9번 찍었습니다 | 반복 |
| 9 | 9 < 10 | 참 | treeHit = treeHit + 1<br>나무를 10번 찍었습니다<br>나무 넘어갑니다 | 반복 |
| 10 | 10 < 10 | 거짓 | | 종료 |

# II. while문

## ❖ while문 만들기

- ✓ 여러 가지 선택지 중 하나를 선택해서 입력 받는 예제
- ✓ 여러 줄의 문자열을 변수에 대입하기 위해 큰따옴표 3개(""")를 이용함

```
>>> prompt = """
...     1.Add
...     2.Del
...     3.List
...     4.Quit
...
...     Enter number: """
```

# II. while문

- ✓ number라는 변수에 0을 먼저 대입함
- ✓ 이렇게 변수를 먼저 설정해 놓지 않으면 다음에 나올 while문의 조건문에서 number 변수가 존재하지 않는다는 에러가 발생함

```
>>> number = 0          ← 번호를 입력 받을 변수
>>> while number !=4:   ← 입력 받은 번호가 4가 아닌 동안 반복한다
...         print(prompt)
...         number = int(input())
...
1.Add
2.Del
3.List
4.Quit

Enter number:
```

- 조건문의 number가 4가 아닌 동안 prompt를 출력하고 번호를 입력 받음
- input()은 사용자가 입력한 문장을 받아들이는 내장 함수임

✓ 결과 화면처럼 사용자가 4라는 값을 입력하지 않으면 계속해서 prompt를 출력함

```
Enter number:
1        ←—— 1 입력


1.Add
2.Del
3.List
4.Quit
```

✓ 4를 입력하면 조건문이 거짓이 되어 while문을 빠져나가게 됨

```
Enter number:
4        ←—— 4 입력
>>>
```

## ❖ while문 강제로 빠져나가기 (break)

✓ while문은 조건문이 참인 동안 계속해서 while문 안의 내용을 반복적으로 수행하지만 강제로 while문을 빠져나가고 싶을 때 사용하는 것이 break문임

✓ 커피 자판기 이야기를 break문으로 만든 예

```
>>> coffee = 10          ← 자판기에 커피가 10개 있음
>>> money = 300          ← 자판기에 넣을 돈은 300원임
>>> while money:
...         print("돈을 받았으니 커피를 줍니다.")
...         coffee = coffee -1          ← while문을 한 번 돌 때 커피가 하나 줄어든다
...         print("남은 커피의 양은 %d개입니다." % coffee)
...         if coffee == 0:
...             print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
...             break
...
```

- 만약 coffee가 0이 되면 if coffee==0: 문장에서 참이 되므로 if문 다음 문장이 수행되고 break문이 호출되어 while문을 빠져나가게 됨

✓ 실제 자판기 작동 과정을 만든 예

```
# coffee.py
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 %d개입니다." % coffee)

    if coffee == 0:
        print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
        break
```

- money = int(input("돈을 넣어 주세요: ") 문장은 사용자로부터 값을 입력 받는 부분이고 입력 받은 숫자를 money 변수에 대입하는 것임

# II. while문

✓ coffee.py 파일을 저장한 후 프로그램을 직접 실행함

```
C:₩doit>python coffee.py
돈을 넣어 주세요:
```

✓ 입력란에 여러 숫자를 입력해 보면서 결과를 확인함

```
돈을 넣어 주세요: 500
거스름돈 200를 주고 커피를 줍니다.
돈을 넣어 주세요: 300
커피를 줍니다.
돈을 넣어 주세요: 100
돈을 다시 돌려주고 커피를 주지 않습니다.
남은 커피의 양은 8개입니다.
돈을 넣어 주세요:
```

## ❖ while문의 맨 처음으로 돌아가기 (continue)

✓ while문을 빠져나가지 않고, while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶을 때 사용하는 것이 continue문임

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

→ a를 2로 나누었을 때 나머지가 0이면 맨 처음으로 돌아감

- a가 짝수이면 continue 문장을 수행하고, while문의 맨 처음으로 돌아감

## ❖ 무한 루프

- ✓ 무한 루프(Loop)란 무한히 반복한다는 의미임
- ✓ while문의 조건문이 True이면 항상 참이므로, while문 안에 있는 문장들은 무한하게 수행됨

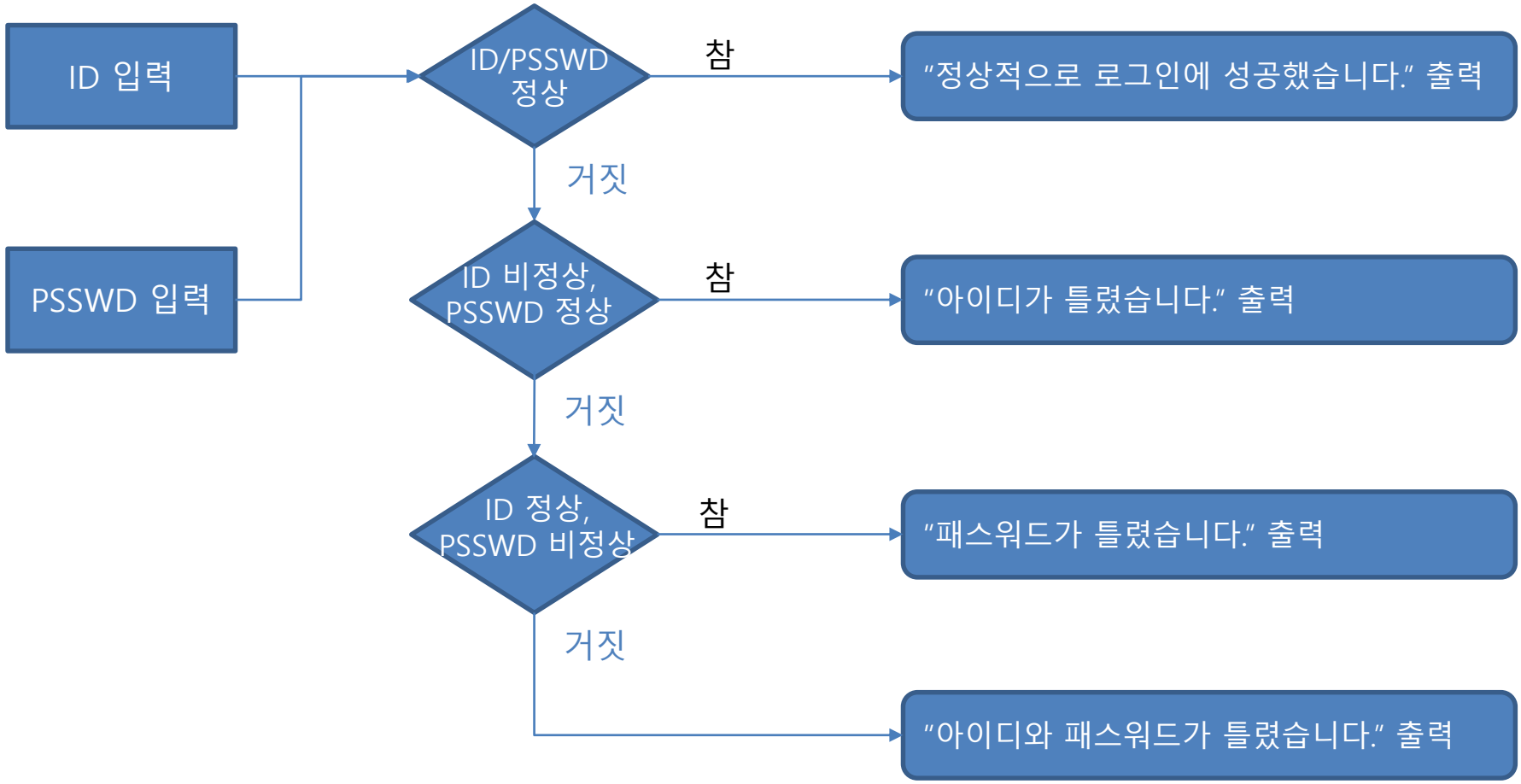
```
while True:
    수행할 문장1
    수행할 문장2
    ...
```

- ✓ 무한 루프의 예

```
>>> while True:
...         print("CTRL+C를 눌러야 while문을 빠져나갈 수 있습니다.")
...
Ctrl+C를 눌러야 while문을 빠져 나갈 수 있습니다.
Ctrl+C를 눌러야 while문을 빠져 나갈 수 있습니다.
Ctrl+C를 눌러야 while문을 빠져 나갈 수 있습니다.
...
```

## 응용예제01. 로그인 프로그램

✓ 아이디와 패스워드를 입력하였을 때, 등록된 정보와 비교하여 로그인을 승인하는 프로그램을 구현해 보자.



## 응용예제02. 별 모양 출력하기

✓ 사용자가 숫자를 여러 개 입력하면 별 모양('₩u2605)을 입력한 숫자만큼 출력하는 프로그램이다.

**1. 입력 받는 값은?**

숫자를 여러 개 입력하세요 : 3628

**2. 출력하는 값은?**

★★★
★★★★★★
★★
★★★★★★★★★

## ❖ for문의 기본 구조

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 수행할 문장들이 수행됨

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

### 1. 전형적인 for문

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

✓ 리스트의 요소들이 차례로 i변수에 대입된 후 print(i)라는 문장을 수행함

## 2. 다양한 for문의 사용

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

✓ a 리스트의 요소값이 튜플이기 때문에 각각의 요소들이 자동으로 (first, last) 변수에 대입됨

## 3. for문의 응용

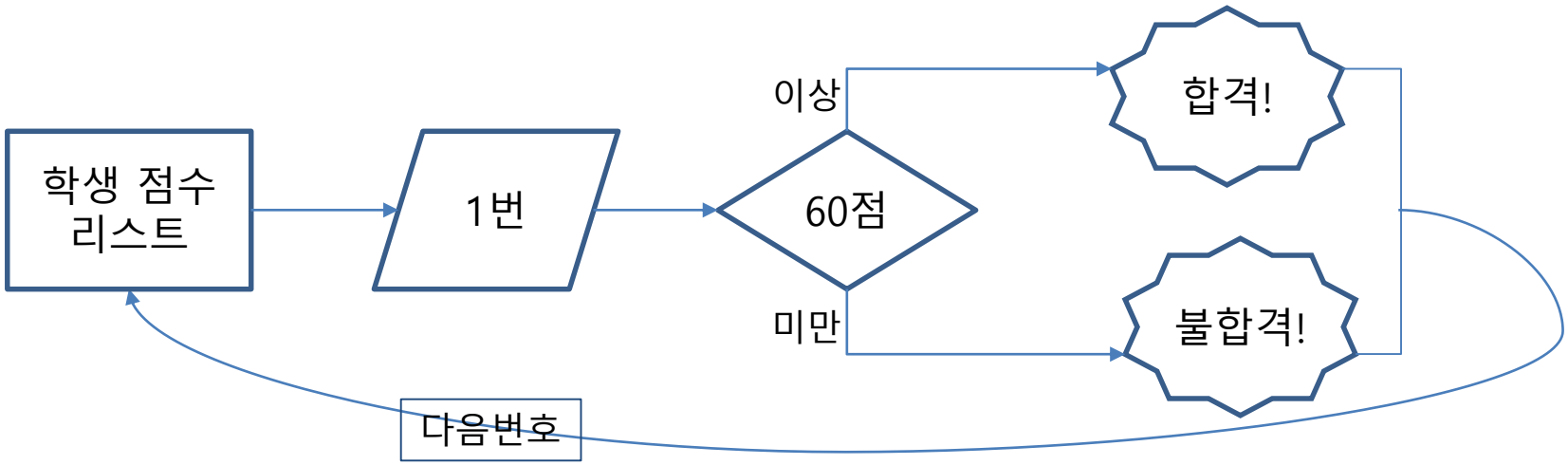
총 5명의 학생이 시험을 보았는데 시험 점수가 60점이 넘으면 합격이고 그렇지 않으면 불합격이다. 합격인지 불합격인지 결과를 보여주시오.

✓ 우선 학생 5명의 시험 점수를 리스트로 표현함

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

✓ 1번 학생은 90점이고, 5번 학생은 80점임

✓ 이런 점수를 차례로 검사해서 합격했는지 불합격했는지 통보해주는 프로그램임



```
#marks1.py
marks = [90, 25, 67, 45, 80]          ← 학생들의 시험 점수 리스트

number = 0          ← 학생에게 붙여 줄 번호
for mark in marks:          ← 90, 25, 67, 45, 80을 순서대로 mark에 대입
    number = number +1
    if mark >= 60:
        print("%d번 학생은 합격입니다." % number)
    else:
        print("%d번 학생은 불합격입니다." % number)
```

✓ 프로그램을 실행하면 mark가 60이상일 때 합격 메시지를 출력하고, 60을 넘지 않을 때 불합격 메시지를 출력함

```
C:₩python>python marks1.py
1번 학생은 합격입니다.
2번 학생은 불합격입니다.
3번 학생은 합격입니다.
4번 학생은 불합격입니다.
5번 학생은 합격입니다.
```

## ❖ for문과 continue

- ✓ for문 안의 문장을 수행하는 도중에 continue문을 만나면 for문의 처음으로 돌아감
- ✓ for문 응용예제를 이용해서 60점 이상인 사람에게는 축하 메시지를 보내고 나머지 사람에게는 아무런 메시지도 전하지 않는 프로그램임

```
#marks2.py
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)
```

- ✓ 점수가 60점 이하인 학생일 경우에는 조건문이 참이 되어 continue문이 수행되고, 축하 메시지를 출력하는 부분인 print문을 수행하지 않고 for문의 처음으로 돌아감

```
C:₩python>python mark2.py
1번 학생 축하합니다. 합격입니다.
3번 학생 축하합니다. 합격입니다
5번 학생 축하합니다. 합격입니다
```

## ❖ for와 함께 자주 사용하는 range함수

✓ for문은 숫자를 자동으로 만들어 주는 range라는 함수와 함께 자주 사용함

```
>>> a = range(10)
>>> a
range(0, 10)      ←—— 0, 1, 2, 3, 4, 5, 6, 7, 8, 9
```

✓ range(10)은 0부터 10미만의 숫자를 포함하는 range 객체를 만들어 줌

✓ 시작 숫자와 끝 숫자를 지정하려면 **range(시작 숫자, 끝 숫자)** 형태를 사용하는데, 이때 끝 숫자는 포함되지 않음

```
>>> a = range(1, 11)
>>> a
range(1, 11)      ←—— 1, 2, 3, 4, 5, 6, 7, 8, 9, 10
```

- **range 함수의 예시 살펴보기**
  - ✓ for와 range 함수를 이용하여 1부터 10까지 더하는 것을 구현함

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

  - range(1, 11)은 숫자 1부터 10까지(1이상 11미만)를 데이터로 갖는 객체임

✓ '60점 이상이면 합격'이라는 문장을 출력하는 예제를 range 함수로 구현함

```
#marks3.py
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 함수는 리스트 안의 요소 개수를 돌려주는 함수임
- len(marks)는 5이고, range(len(marks))는 range(5)임
- number 변수는 0부터 4까지 숫자가 대입될 것이고, marks[number]는 차례대로 점수 값을 갖게 됨

- **for와 range를 사용한 구구단**

```
>>> for i in range(2, 10):
...         for j in range(1, 10):
...             print(i*j, end=" ")     ← 출력할 때 다음 줄로 넘기지 않고 그 줄에 계속 출력함
...         print(' ')    ← 출력할 때 다음 줄로 넘겨서 출력함
...
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
```

- ✓ 첫 번째 for문은 2부터 9까지의 숫자(range(2,10))가 차례로 i에 대입됨
- ✓ 두 번째 for문은 1부터 9까지의 숫자(range(1, 10))가 차례로 j에 대입됨
- ✓ 그 다음 문장 print(i*j)를 수행함

## ❖ 리스트 내포 사용하기

리스트 안에 for문을 포함하는 리스트 내포(List comprehension)를 사용하면 편리함

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

✓ a 리스트의 각 항목에 3을 곱한 결과를 result 리스트에 담는 예제임

✓ 리스트 내포를 사용하면 간단히 해결할 수 있음

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

✓ 짝수에만 3을 곱하여 담고 싶다면 리스트 내포 안에 'if 조건'을 사용함

```
>>> a = [1, 2, 3, 4]
>>> result = [num * 3 for num in a if num % 2 == 0]
>>> print(result)
[6, 12]
```

✓ **리스트 내포의 일반적인 문법**

[표현식 **for** 항목 **in** 반복 가능 객체 **if** 조건]

✓ **for문을 여러 개 사용할 때의 문법**

```
[표현식  for 항목 in 반복 가능 객체 if 조건1
         for 항목 in 반복 가능 객체 if 조건2
         ...
         for 항목 in 반복 가능 객체 if 조건n]
```

✓ 구구단의 모든 결과를 리스트에 담고 싶다면 리스트 내포를 사용하여 구현함

```
>>> 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]
```