3."R{vjqp"< 제어문 *Eqpvtqn"Uvcvg o gpv+

- Ch3 에서는 if, while, for 등의 제어문에 대해서 배워본다. 이전 장에서 배운 자료형을 바탕으로 제어문을 통해서 프로그램의 구조를 살펴보자.

Table of Contents

- Tips : 제어문을 다루기 전에 다음의 사항을 미리 확인해 보자!

  - 제어문에 속하는 모든 문장은 반드시 들여쓰기를 해 주어야 함(들여쓰기를 하지 않는 경우 오류 발생)
  - 제어문에 속하는 모든 문장들의 들여쓰기 너비가 동일해야 함(동일하지 않은 경우 오류 발생)
  - 들여쓰기는 TAB 한 번 또는 스페이스 4칸으로 사용하는데, 탭은 환경에 따라 공백 수가 다를 수 있어서 스페이스로 사용하는 추세

1. if 조건문

- if 조건문은 조건을 판별할 때 사용하는 문장을 말한다.
- 조건의 참, 거짓 여부에 따라 프로그램 제어 또는 반복하는 수행을 할 수 있다.

if 문의 기본 구조

if 조건문:

    수행할 문장1
    수행할 문장2
    ...

elif 조건문:

    수행할 문장1
    수행할 문장2
    ...

else:

    수행할 문장1
    수행할 문장2
    ...

In [5]:
```
# example : if 조건이 참 (True)이 되면 문장을 수행한다.
money = False
if money:
   print("지금 사자")
else :
   print("나중에 사자")
```

나중에 사자

In [6]:
```
# example : if 조건이 거짓(False)가 되면 else 다음의 문장을 수행한다.
money = False
if money: print("지금 사자")
else : print("나중에 사자")
```

나중에 사자

비교 연산자

- 조건문에서 조건의 만족 여부를 판단하기 위해 사용되는 비교연산자를 사용하는 방법을 알아보자.

비교 연산자	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

In [7]:
```
x = 200
y = 100
x > y # 해당 조건은 True
```

Out[7]: True

In [8]:
```
x == y # 해당 조건은 False
```

Out[8]: False

아래와 같이 if elif else 를 이용한 조건문을 확인해 보자.

In [9]:
```
# 표현 1
if x > y:
    print('x가 y보다 크다.')
elif x<y:
    print('x가 y보다 작다.')
else:
    print('x가 y가 같다.')
```

x가 y보다 크다.

동일한 내용을 아래와 같이 표현 할수 있다.

In [24]:
```
# 표현 2
if x > y:
    print('x가 y보다 크다.')
else:
    if x<y:
        print('x가 y보다 작다.')
    else:
        print('x가 y가 같다.')
```

x가 y보다 크다.

이번엔 if else 문을 확인해 보자.

In [16]:
```
# if else 문 : 표현1
score=75
if score>=60:
    message = "success"
else:
    message = "failure"
print(message)
```

success

위와 동일한 내용을 아래와 같이 조건문 표현식 (conditional expression)을 사용하여 간단히 표현 할 수 있다.

In [17]:
```
# if else 문 : 표현2
score=75
```

```
message = "success" if score>=60 else "failure"
print(message)
```

success

조건부 표현식은 다음과 같이 정의한다.

(조건문이 참인 경우) if (조건문) else (조건문이 거짓인 경우)

조건부 표현식은 한 줄로 작성 할 수 있기 때문에 가독성이 좋다.

and, or, not 사용

- 조건을 판단하기 위해 사용하는 다른 연산자로는 and, or, not이 있다. 각각의 연산자는 다음처럼 동작한다.

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

In [18]:
```
money = 2000
card = True
if money >= 3000 or card:
    print("지금 사자!")
else:
    print("나중에 사자")
```

지금 사자!

In [19]:
```
money = 2000
if not money < 3000 :
    print("지금 사자!")
else:
    print("나중에 사자")
```

나중에 사자

in, not in 사용

- 파이썬에서는 in 사용하여 조건문을 만들 수 있는데, in 의 뜻이 "~안에" 라는 것을 생각해 보면, 그 의미를 이해할 수 있다.
- x in s : s 안에 x 가 있으면 True/ 없으면 False를 반환한다.
- x not in s : s 안에 x 가 없으면 True / 있으면 False를 반환한다.
- s 는 리스트, 튜플, 문자열을 사용할 수 있다.

In [20]:
```
#s : 리스트
100 in [100, 200, 300]
```

Out[20]: True

In [21]:
```
#s : 튜플
150 not in (100, 200, 300)
```

Out[21]: True

In [22]:
```
#s : 문자열
'j' not in 'python'
```

Out[22]: True

아래와 같이 in , not in 을 사용해서 if 조건문을 만들어보자.

In [23]:
```
lst = [200, 300, 400]
if 100 in lst:
    print('100이 리스트 안에 존재합니다.')
else:
    print('100이 리스트 안에 존재하지 않습니다.')
```

100이 리스트 안에 존재하지 않습니다.

In [24]:
```
tup = (200, 300, 400)
if 100 in tup:
    print('100이 튜플 안에 존재합니다.')
elif 100 not in tup:
    print('100이 튜플 안에 존재하지 않습니다.')
```

100이 튜플 안에 존재하지 않습니다.

In [25]:
```
st = "student"
if "s" in st:
    print('s가 문자열 안에 존재합니다.')
else:
    print('s가 문자열 안에 존재하지 않습니다.')
```

s가 문자열 안에 존재합니다.

- Tip : 만일 조건문에서 어떠한 수행도 하지 않고, 넘어가게 설정하고 싶은 경우 pass 를 사용한다.

In [26]:
```
pocket = ['paper', 'money', 'cellphone']
if 'money' in pocket: pass
else: print("카드를 꺼내라")
```

--> pocket 리스트 안에 money 문자열이 있기 때문에 if문 다음 문장인 pass가 수행되고 아무 결괏값도 보여 주지 않는다.

2. while 반복문

- 반복해서 문장을 수행해야 할 경우 while문을 사용한다.

- 다음은 while 반복문의 기본구조이다.

while 조건문:

    수행할 문장1
    수행할 문장2
    수행할 문장3
    ...

while 문 만들기

while문은 조건문이 참인 동안에 while문 아래의 수행할 문장이 반복해서 수행된다.

In [27]:
```
# i < 5 인 동안에 i 를 출력하고 i+1 로 i를 재정의 함
i=0
while i<5:
  print(i)
  i = i+1   # i+=1  도 동일한 표현
```

```
0
1
2
3
4
```

In [28]:
```
# treeHit <10 인 동안에 treeHit을 하나씩 키우면서, 첫번째 print 내용을 출력하고, treeHit 이 10 이 되면 "나무 넘어갑니다" 출력
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 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.
```

input 함수를 사용하여, 사용자로 부터 입력 받는 프로그램을 만들어 while 문과 if 문을 함께 사

용해보자.

In [29]:
```
number =23
running =True

while running:
   guess = int(input('Enter an integer : '))
   if guess == number:
      print ('Congratulations, you guessed it.')
      # this causes the while loop to stop
      running = False
   elif guess < number:
      print ('No, it is a little higher than that.')
   else:
      print ('No, it is a little lower than that.')
```

```
Enter an integer : 15
No, it is a little higher than that.
Enter an integer : 25
No, it is a little lower than that.
Enter an integer : 23
Congratulations, you guessed it.
```

while 문 강제로 빠져 나가기

- while문을 강제로 종료하고 싶은 경우 break 사용한다.

In [30]:
```
i = 1
while i:
   print(i)
   i += 1
   if i>=5: # i가 5 이상이면 while 문을 강제로 종료한다.
      break
```

```
1
2
3
4
```

위 예시에서 만일 if문과 break가 쓰이지 않았다면 while문은 무한히 반복되게 된다. (0아 아닌 i는 계속 참이기 때문에)

이번엔 while 문을 이용해서, 아래와 같이 커피 자판기를 상상할 수 있는 코드를 작성해 보자.

In [36]:
```
coffee = 5
money = 100
while money:
   print("돈을 받았으니 커피를 줍니다.")
   coffee = coffee-1 # 또는 coffee -= 1
   print("남은 커피의 양은 %d개입니다." % coffee)
   if coffee == 0:
      print("커피가 다 떨어졌습니다. 판매를 중지합니다.")
```

```
    break
```

```
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 4개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 3개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 2개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 1개입니다.
돈을 받았으니 커피를 줍니다.
남은 커피의 양은 0개입니다.
커피가 다 떨어졌습니다. 판매를 중지합니다.
```

위의 반복문은 money가 100으로 고정되어 있으므로

while money:에서 조건문인 money는 0이 아니기 때문에 항상 참이다.

따라서 무한히 반복되는 무한 루프를 돌게 된다. break 문장이 없다면 while 문을 빠져나올 수 없게 된다.

아래는 자판기의 실제 작동 과정과 비슷하게 만들어본 예이다.

money = int(input("돈을 넣어 주세요: ")) 문장은 사용자로부터 값을 입력받는 부분이고 입력받은 숫자를 money 변수에 대입하는 것이다.

In [37]:
```
coffee = 5
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
```

```
돈을 넣어 주세요: 400
거스름돈 100를 주고 커피를 줍니다.
돈을 넣어 주세요: 300
커피를 줍니다.
돈을 넣어 주세요: 500
거스름돈 200를 주고 커피를 줍니다.
돈을 넣어 주세요: 300
커피를 줍니다.
돈을 넣어 주세요: 500
거스름돈 200를 주고 커피를 줍니다.
커피가 다 떨어졌습니다. 판매를 중지 합니다.
```

while문의 맨 처음으로 돌아가기

- while문 안의 문장을 수행할 때 입력 조건을 검사해서 조건에 맞지 않으면 보통 while문을 빠져나간다.
- while문을 빠져나가지 않고 while문의 맨 처음(조건문)으로 다시 돌아가게 만들고 싶은 경우 사용하는 것이 바로 continue문이다.
- 먼저 살펴본 pass는 단순히 실행할 코드가 없다는 것을 나타내는 명시적인 역할을 하고, continue는 다음 반복루프로 넘어가게 하는 역할을 한다.

아래와 같이 continue 를 활용해서, 1부터 10까지의 숫자 중에서 홀수만 출력하는 while 문을 작성해 보자.

In [38]:
```
a = 0
while a<10:
    a = a+1
    if a%2 == 0: continue # a<10이고 짝수인 경우 print(a)를 수행하지않고, 처음으로 돌아감
    print(a)
```

```
1
3
5
7
9
```

3. for 반복문

- while문과 비슷한 반복문인 for문은 매우 유용하고 문장 구조가 한눈에 쏙 들어온다는 장점이 있다.

for 문의 기본 구조는 아래와 같다.

for 변수 in 리스트(또는 튜플, 문자열):

    수행할 문장1
    수행할 문장2
    ...

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

다양한 for 문의 예시를 확인해보자.

In [7]:
```
# 리스트 활용
for x in [1, 2, 3]: # 리스트의 요소들을 하나씩 x에 대입
    print(x)
```

```
test_list = ['one', 'two', 'three'] # 변수를 지정해서 for의 조건으로 사용 가능
for i in test_list:
    print(i)
```

```
1
2
3
one
two
three
```

In [9]:
```
# 튜플 활용
for x in (1, 2, 3):
    print(x)
```

```
1
2
3
```

In [11]:
```
# 문자열 활용
for x in 'abcde': # 문자열
    print(x)
```

```
a
b
c
d
e
```

In [45]:
```
# 딕셔너리 활용
dic = {'key1':1, 'key2':2, 'key3':3} # 딕셔너리
for x in dic:
    print(x, ":", dic[x])
```

```
key1 : 1
key2 : 2
key3 : 3
```

다양한 for 문

튜플을 요소로 같는 리스트를 활용하여 아래와 같은 for문을 만들어보자.

In [46]:
```
a = [(1,2), (3,4), (5,6)]
for (first, last) in a:   # a 리스트의 요소값은 튜플 --> 튜플의 각 요소가 first, last로 대입됨
    print(first + last)
```

```
3
7
11
```

아래와 같이 for문 안에 if 문을 사용 할 수 있다.

In [47]:
```
for i in [1, 2, 3, 4, 5]:
```

```
  if i%2 == 0:
    print(i, '는 2의 배수')
  elif i%3 == 0:
    print(i, '는 3의 배수')
  else:
    pass  # 수행 없음
```

2 는 2의 배수
3 는 3의 배수
4 는 2의 배수

while 문에서 살펴본 continue 문을 for문에서도 사용할 수 있다. 문장 수행도중에 continue 문을 만나면 for문의 처음으로 돌아간다.

In [48]:
```
for i in [1, 2, 3, 4, 5]:
  if i % 2 == 0:
    print(i, '는 2의 배수')
  elif i % 3 == 0:
    continue
    print(i, '는 3의 배수') # 이 명령은 실행되지 않음
  else:
    pass
```

2 는 2의 배수
4 는 2의 배수

range 함수 사용

- for문은 숫자 리스트를 자동으로 만들어 주는 range 함수와 함께 사용하는 경우가 많다. 다음은 range 함수의 간단한 사용법이다.

In [49]:
```
# range(11)은 0이상 11 미만의 숫자를 포함하는 객체를 만들어 준다.
a = range(11)
a
```

Out[49]: range(0, 11)

In [50]:
```
# 시작 숫자와 끝 숫자를 지정하려면 range(시작 숫자, 끝 숫자) 형태를 사용한다. (0이상 11미만의 숫자를 포함하는 객체)
b = range(0, 11)
```

In [51]:
```
b
```

Out[51]: range(0, 11)

In [52]:
```
a == b
```

Out[52]: True

In [53]:
```
c=range(5,11)
c
```

Out[53]:

range(5, 11)

In [40]:
```
for i in range(6):
    print(i)
```

```
0
1
2
3
4
5
```

for 문에서 사용되는 range 함수의 예시를 좀 더 살펴보자

for와 range 함수를 사용하면 1부터 10까지 더하는 것을 다음과 같이 쉽게 구현할 수 있다.

In [54]:
```
add = 0
for i in range(1, 11):
    add = add+i   # 1 이상 11 미만의 숫자가 이전 add에 계속 더해짐
print(add)
```

```
55
```

for문 안에는 break와 continue 를 사용 할 수있다.

In [55]:
```
for x in [1, 2, 3]:
    print(x)
    if x==2: # x가 2일 때 반복문 종료
        break
```

```
1
2
```

In [56]:
```
for x in range(1, 11):
    if x%2 == 1: # x가 홀수 이면 처음으로 돌아감
        continue
    print(x) # 짝수인 경우만 출력됨
```

```
2
4
6
8
10
```

for문 중첩 사용

- for와 range 함수를 사용하면 소스 코드 단 4줄만으로 구구단을 출력할 수 있다.

In [42]:
```
for i in range(2, 10): # ①번 for문
    for j in range(1, 10): # ②번 for문
        print(i*j, end=" ") # end를 넣어 준 이유는 해당 결괏값을 출력할 때 다음줄로 넘기지 않고
```

```
그 줄에 계속해서 출력하기 위해
    print('') # 2단, 3단 등을 구분하기 위해 두 번째 for문이 끝나면 결괏값을 다음 줄부터 출력하게 해주는 문장
```

```
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
```

리스트 내포 (List comprehension) 사용

- 리스트 안에 for문을 포함하는 리스트 내포(List comprehension)를 사용하면 좀 더 편리하고 직관적인 프로그램을 만들 수 있다.

- 리스트 내포의 일반 문법

  [표현식 for 항목 in 반복가능객체 if 조건문]

- for문을 두개 사용한 리스트 내포의 문법

  [표현식 for 항목1 in 반복가능객체1 if 조건문1

  for 항목2 in 반복가능객체2 if 조건문2
  ...
  for 항목n in 반복가능객체n if 조건문n]

위 리스트 내포의 문법에서 if 조건문 부분은 생략이 가능하다.

아래 예제는 a 리스트의 각 항목에 3을 곱한 결과를 result 리스트에 담는 예제이다.

In [57]:
```
a = [1, 2, 3, 4]
result = []

for num in a:
    result.append(num*3)
print(result)
```

```
[3, 6, 9, 12]
```

위 5줄 자리 코드는 리스트 내포를 사용하면 간단하게 표현 할 수 있다.

In [58]:
```
a = [1, 2, 3, 4]
result = [num*3 for num in a]
print(result)
```

[3, 6, 9, 12]

만약 [1,2,3,4] 중에서 짝수에만 3을 곱하여 담고 싶다면 다음과 같이 리스트 내포 안에 "if 조건"을 사용할 수 있다.

In [59]:
```
a = [1, 2, 3, 4]
result = [num*3 for num in a if num%2 == 0] # 짝수일 때만 수행
print(result)
```

[6, 12]

만약 구구단의 모든 결과를 리스트에 담고 싶다면 리스트 내포를 사용하여 다음과 같이 for 문을 두개 사용하여 간단하게 구현할 수도 있다.

In [45]:
```
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]

Example

Q1) while문을 사용하여 다음과 같이 아래 십자가(+)을 표시하는 프로그램을 작성해 보자.

In [58]:
```
Image("fig1.jpg")
```

Out[58]:

Q2) for문을 사용해 1부터 20까지의 숫자를 출력해 보자.

Q3) [1, 2, 3, 4, 5] 리스트 중에서 홀수에만 3을 곱해서 저장하는 코드를 리스트 내포를 사용해서 작성해 보자.

Q4) while문을 사용해 1부터 1000까지의 자연수 중 3의 배수의 합을 구해 보자.

Q5) 리스트 내포 기능을 이용해 다음 문장으로부터 모음('aeiou')을 제거해 보자.
'Python is powerfuland fast; plays well with others'