

1장 기초 수학 프로그램



1.1 세 정수의 정렬



프로그램 개요

- 세 개의 정수를 입력 받아서 이것을 순서대로 정렬하는 프로그램을 작성

- 입력 : 세 개의 정수 a, b, c
- 출력 : 오름차순으로 정렬된 세 정수

- 예) $a = 3, b = 1, c = 2$
 - 입력 받은 세 정수를 정렬하여 1, 2, 3을 출력

파이썬 문법



변수

- 변수(variable)
 - 값을 저장하는 기억장소를 가리키는 이름
 - 변수에 저장되는 값은 수시로 변할 수 있음
- 변수명을 정하는 규칙
 - 변수명은 영문자, 숫자, 밑줄 기호(_)만을 사용함
 - 밑줄 기호(_)를 제외한 다른 기호는 사용할 수 없음
 - 첫 글자는 반드시 영문자 또는 밑줄 기호(_)로 시작해야 함 (숫자는 안 됨)
 - 대소문자를 구분함 (name, Name, NAME은 모두 다른 이름으로 간주)
 - 다음과 같은 키워드(예약어)를 사용하지 않음
 - False, None, True, and, as, asser, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield

치환문 (1)

- 치환문의 형식

```
변수 = 값  
변수1 = 변수2
```

- a라는 변수에 정수 3을 저장할 때

```
>>> a = 3  
>>> a  
3
```

치환문 (2)

- 변수에는 정수 외에도 다음과 같이 실수나 문자열 등의 다양한 값을 저장할 수 있음

```
>>> a = 3.1
>>> a
3.1
>>> a = 'string'
>>> a
'string'
```

- 하나의 변수에 있는 값을 다른 변수에 저장할 때

```
>>> a = 10
>>> b = a
>>> a
10
>>> b
10
```

치환문 (3)

- 변수에 계산 결과를 저장할 때

```
>>> a = (1+3)*4-5  
>>> a  
11
```

- 변수 a의 값과 변수 b의 값을 교환할 때

```
>>> a = 1  
>>> b = 2  
>>> a, b = b, a  
>>> a  
2  
>>> b  
1
```


print() 함수 (1)

- print()는 매개변수를 출력하는데 사용되는 함수
 - 매개변수로 값이나 문자열이 입력되면 그대로 화면에 출력
 - 매개변수로 변수가 입력되면 변수의 값을 화면에 출력

```
>>> print(3)
3
>>> print('a = ')
a =
>>> a = 3
>>> print(a)
3
>>> print('a = ', a)
a = 3
```

print() 함수 (2)

- 중간에 줄바꿈을 하고 싶으면 줄바꿈을 나타내는 문자 '\n'을 넣어 줌 (한글 키보드에서는 ' \' 가 '\n'임)

```
>>> print(1, '\n', 2, 3, '\n', 4, 5, 6, '\n', 7, 8, 9, 10)
1
2 3
4 5 6
7 8 9 10
```

- 변수를 하나만 출력할 때 다음과 같이 변수 이름을 써 주고 엔터 키를 누르는 것은 print() 함수를 사용하는 것과 결과가 동일 (변수에 정수 값이 저장된 경우)

```
>>> a = 3
>>> a
3
>>> print(a)
3
```

print() 함수 (3)

- 여러 개의 변수 값을 출력할 때에는 반드시 print() 함수를 사용해야 함

```
>>> a = 1
>>> b = 2
>>> c = 3
>>> print(a, b, c)
1 2 3
```

- 여러 개의 변수를 콤마로 연결한 다음 엔터 키를 누르면 튜플(tuple) 자료형으로 저장됨

```
>>> a, b, c
(1, 2, 3)
```

input() 함수 (1)

- input()은 데이터를 입력 받는 함수
 - 입력 받은 데이터는 문자열로 저장됨

```
>>> a = input()
```

```
3
```

```
>>> a
```

```
'3'
```

```
>>> b = input()
```

```
3.7
```

```
>>> b
```

```
'3.7'
```

```
>>> c = input()
```

```
홍길동
```

```
>>> c
```

```
'홍길동'
```

input() 함수 (2)

- 데이터를 입력하기 전에 설명문을 출력할 때

```
>>> a = input('학년 : ')
학년 : 3
>>> b = input('학점 : ')
학점 : 3.7
>>> c = input('이름 : ')
이름 : 홍길동
>>> a
'3'
>>> b
'3.7'
>>> c
'홍길동'
```

int() 함수 (1)

- int()는 매개변수로 입력 받은 문자열을 정수로 변환해 주는 함수
- 변수에 직접 값을 저장한 경우와 input() 함수를 통해 값을 입력 받은 경우의 차이

```
>>> a = 3
>>> b = 4
>>> a + b
7
>>> a = input()
3
>>> b = input()
4
>>> a + b
'34'
```

int() 함수 (2)

- input()을 통해 입력 받은 문자열을 정수로 변환하기 위해서는 int() 함수를 사용해야 함

```
>>> a = int(input('a = '))  
a = 30  
>>> a  
30
```

- input() 함수만을 사용하여 데이터를 입력 받으면 다음과 같이 문자열로 저장됨

```
>>> b = input('b = ')  
b = 30  
>>> b  
'30'
```

int() 함수 (3)

- 정수 외에 다른 수나 문자를 입력하면 다음과 같이 오류가 발생함

```
>>> a = int(input('a = '))
a = 1.1
Traceback (most recent call last):
  File "<pyshell#9>", line 1, in <module>
    a = int(input('a = '))
ValueError: invalid literal for int() with base 10: '1.1'
>>>
>>> a = int(input('a = '))
a = b
Traceback (most recent call last):
  File "<pyshell#11>", line 1, in <module>
    a = int(input('a = '))
ValueError: invalid literal for int() with base 10: 'b'
```


프로그래밍 연습 (1)

- 자신의 학과와 이름을 입력으로 받아, 다음 예와 출력하는 프로그램을 작성

```
>>>
```

```
학과 : 컴퓨터공학부
```

```
이름 : 홍길동
```

```
저는 컴퓨터공학부 에 재학중인 홍길동 입니다.
```

```
>>>
```

프로그래밍 연습 (2)

- 마일을 입력으로 받아 킬로미터로 변환해서 출력하는 프로그램을 작성
 - 킬로미터 = 마일 * 1.609344

```
>>>
```

```
마일 : 1
```

```
킬로미터 : 1.609344
```

```
>>>
```

```
마일 : 20
```

```
킬로미터 : 32.18688
```

```
>>>
```

```
마일 : 60
```

```
킬로미터 : 96.56064
```

```
>>>
```

> (관계연산자)

- 관계 연산자 >은 첫 번째 피연산자가 두 번째 피연산자보다 크면 참(True)을 반환하고, 작거나 같으면 거짓(False)을 반환함

```
>>> 5 > 3
True
>>> 5 > 5
False
>>> 5 > 6
False
```

- 크기를 비교하는 관계 연산자
 - > (크다), < (작다), >= (크거나 같다), <= (작거나 같다)

if 문 (1)

- if문의 구조

```
if 조건: 명령문1  
else: 명령문2
```

- if문의 동작 원리

- 사용자가 지정한 조건을 만족하면 명령문1이 실행됨
- 조건을 만족하지 못하면 명령문2가 실행됨
- else문은 필요한 경우에만 사용함

if 문 (2)

- <프로그램 1.1> 기온에 따라 날씨가 더운지 여부 출력

```
t = int(input('기온 : '))  
if t > 30: print('날씨가 덥다.')  
else: print('날씨가 덥지 않다.')
```

- <프로그램 1.1>의 실행 결과

```
>>>  
기온 : 35  
날씨가 덥다.  
>>>  
기온 : 29  
날씨가 덥지 않다.
```

if 문 (3)

- if문을 사용할 때 하나 이상의 명령문이 필요한 경우에는 명령문을 다음 줄에서 시작해야 하는데, 이 때는 반드시 다음과 같이 들여쓰기를 해야 함

```
t = int(input('기온 : '))  
if t > 30:  
    print('날씨가 덥다.')  
else:  
    print('날씨가 덥지 않다.')
```

- 파이썬에서는 들여쓰기를 제대로 하지 않은 경우 다음과 같은 구문 오류(Syntax Error)가 발생함

```
>>> if t > 30:  
print('날씨가 덥다.')  
SyntaxError: expected an indented block
```

if 문 (4)

- 검사해야 할 조건들이 여러 개 있는 경우에는 다음과 같이 elif를 사용함

```
if 조건1: 명령문1
elif 조건2: 명령문2
elif 조건3: 명령문3
...
else: 명령문n
```

if 문 (5)

- <프로그램 1.2> 여러 개의 조건문 사용

```
t = int(input('기온 : '))
if t > 30:
    print('날씨가 덥다.')
elif t > 20:
    print('날씨가 따뜻하다.')
elif t > 10:
    print('날씨가 시원하다.')
else:
    print('날씨가 춥다.')
```


if 문 (6)

- <프로그램 1.2>의 실행 결과

```
>>>  
기온 : 35  
날씨가 덥다.  
>>>  
기온 : 24  
날씨가 따뜻하다.  
>>>  
기온 : 17  
날씨가 시원하다.  
>>>  
기온 : -10  
날씨가 춥다.
```

if 문 (7)

- if-elif문을 사용하면 여러 개의 조건 중에서 하나만 선택함
- if-elif문을 사용하지 않고, 다음 예와 같이 if문을 여러 개 사용할 경우 실행 결과가 다르게 나옴

```
t = int(input('기온 : '))
if t > 30:
    print('날씨가 덥다.')
if t > 20:
    print('날씨가 따뜻하다.')
if t > 10:
    print('날씨가 시원하다.')
else:
    print('날씨가 춥다.')
```

주석문 (1)

- 주석문은 프로그램에서 실행시키고 싶지 않은 문장이나 프로그램에 대한 설명문을 넣고 싶은 경우 사용
- 편집 창에서 #을 입력하면 # 이후에 나오는 문자가 빨간색으로 변하게 되며, # 이후에 나오는 부분이 주석문으로 처리되어 실행되지 않음

```
t = int(input('기온 : '))
if t > 30:
    print('날씨가 덥다.')
elif t > 20:
    print('날씨가 따뜻하다.')
elif t > 10:
    print('날씨가 시원하다.')
#else:
#    print('날씨가 춥다.')
```

주석문 (2)

- #은 다음과 같이 프로그램에 대한 설명을 넣을 때도 유용하게 사용할 수 있음

```
t = int(input('기온 : '))
if t > 30:                # 기온이 30도 초과인 경우
    print('날씨가 덥다.')
elif t > 20:              # 기온이 20도 초과 30도 이하인 경우
    print('날씨가 따뜻하다.')
elif t > 10:              # 기온이 10도 초과 20도 이하인 경우
    print('날씨가 시원하다.')
else:                     # 그 외의 경우
    print('날씨가 춥다.')
```

주석문 (3)

- 여러 줄을 주석문 처리할 때에는 다음과 같이 큰 따옴표 세 개("""를 사용

```
t = int(input('기온 : '))
if t > 30:          # 기온이 30도 초과인 경우
    print('날씨가 덥다.')
    """

elif t > 20:        # 기온이 20도 초과 30도 이하인 경우
    print('날씨가 따뜻하다.')
elif t > 10:        # 기온이 10도 초과 20도 이하인 경우
    print('날씨가 시원하다.')
    """

else:               # 그 외의 경우
    print('날씨가 춥다.')
```

주석문 (4)



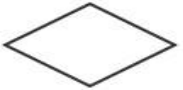


- 처음이나 중간에 프로그램에 대한 설명문을 넣고 싶은 경우에도 큰 따옴표 세 개를 사용함

```
"""
기온에 따라 날씨의 상태를 출력한다.
작성자 : 채진석
작성 일자 : 10월 17일
"""

t = int(input('기온 : '))
if t > 30:                # 온도가 30도 초과인 경우
    print('날씨가 덥다.')
elif t > 20:              # 온도가 20도 초과 30도 이하인 경우
    print('날씨가 따뜻하다.')
elif t > 10:              # 온도가 10도 초과 20도 이하인 경우
    print('날씨가 시원하다.')
else:                     # 그 외의 경우
    print('날씨가 춥다.')
```

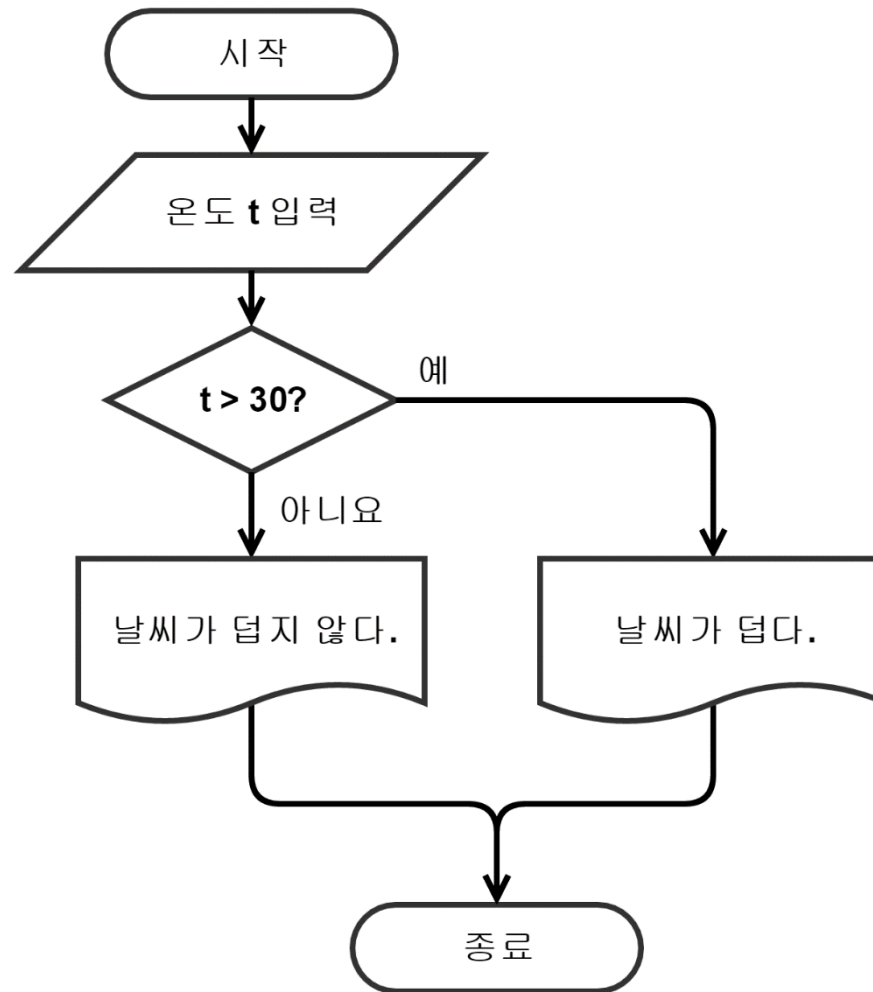
순서도 (1)

- 순서도(flowchart)란?
 - 순서도는 미리 정의된 기호와 그것들을 서로 연결하는 선을 사용하여 프로그래밍의 여러 문제를 해결하는 데 필요한 논리적인 흐름을 그림으로 표현한 것
- 순서도의 기호와 의미

기호	의 미
	순서도의 시작과 끝을 나타내는 기호
	값을 계산하거나 대입 등을 나타내는 기호
	조건이 참이면 '예', 거짓이면 '아니요'로 가는 판단 기호
	데이터 입력에 사용되는 입력 기호
	문서로 인쇄할 것을 나타내는 인쇄 기호

순서도 (2)

- <프로그램 1.1>에 대한 순서도



프로그램 작성

- 세 개의 정수를 입력 받아 오름차순으로 정렬하는 프로그램 작성

```
>>>
정수 a, b, c 입력
a = 3
b = 2
c = 1
정렬 전 : 3 2 1
정렬 후 : 1 2 3
>>>
정수 a, b, c 입력
a = 2
b = 1
c = 3
정렬 전 : 2 1 3
정렬 후 : 1 2 3
>>>
```

프로그래밍 과제

- 네 개의 정수를 입력 받아 오름차순으로 정렬하는 프로그램 작성

```
>>>
정수 a, b, c, d 입력
a = 4
b = 3
c = 2
d = 1
정렬 전 : 4 3 2 1
정렬 후 : 1 2 3 4
>>>
정수 a, b, c, d 입력
a = 2
b = 1
c = 4
d = 3
정렬 전 : 2 1 4 3
정렬 후 : 1 2 3 4
>>>
```

1.2 모든 약수 구하기



프로그램 개요

- 자연수 a 를 입력 받아서 a 의 모든 약수를 출력하는 프로그램을 작성

- 입력 : 자연수 a
- 출력 : a 의 모든 약수

- 약수(divisor)
 - 어떤 수를 나누어 떨어지게 하는 수
 - 1은 모든 수의 약수이고, 어떤 수는 자기 자신의 약수임
- 예) $a = 12$
 - 12의 모든 약수인 1, 2, 3, 4, 6, 12를 출력

파이썬 문법



while문 (1)

- while문의 형식

```
while 조건: 명령문
```

- while문은 반복문으로 조건이 참일 경우에만 다음에 나오는 문장을 반복
- 1부터 5까지 자연수를 출력할 때

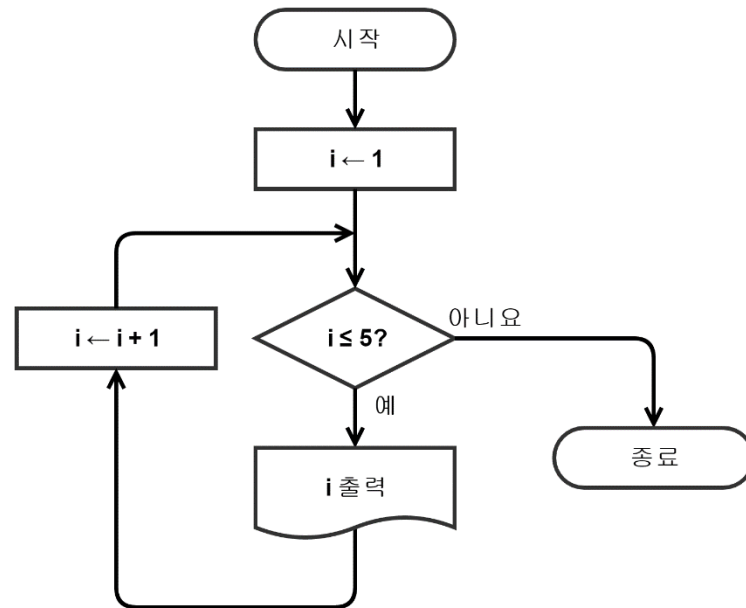
```
i = 1  
while i <= 5:  
    print(i)  
    i = i + 1
```

코드 블록(code block)이라고 부르며,
동일한 위치로 들여쓰기 해야 함

while문 (2)

- while문에 대한 순서도

```
i = 1  
while i <= 5:  
    print(i)  
    i = i + 1
```



while문 (3)

- i 의 값을 1 증가시키는 명령문인 ' $i = i + 1$ '은 다음과 같이 간단하게 나타낼 수 있음

```
i += 1
```

- 빼기, 곱하기, 나누기 연산자인 $-$, $*$, $/$ 에 대해서도 다음과 같이 표현할 수 있음

```
>>> i = 20
>>> i -= 10
>>> i
10
>>> i *= 3
>>> i
30
>>> i /= 2
>>> i
15.0
```

나누기 연산자는 결과 값을 실수로 저장

for문 (1)

- for문의 형식

```
for 변수 in range(시작범위, 끝범위): 명령문
```

- for문은 반복문으로 range()에 있는 범위만큼 다음에 나오는 문장을 반복
- 1부터 5까지 자연수를 출력할 때

```
for i in range(1, 6) print(i)
```

끝범위는 종료하기 원하는 값보다 1 더한 값을 써야 함

for문 (2)

- range() 함수에서 시작범위를 생략하면, 시작범위는 0부터 시작함

```
>>> for i in range(5): print(i)
```

```
0  
1  
2  
3  
4
```

- 모든 for문은 while문을 사용하여 나타낼 수 있음

```
i = 1  
while i < 6:  
    print(i)  
    i += 1
```

→ "for i in range(1, 6): print(i)"와 동일

end (매개변수)

- print() 함수에서 줄바꿈을 하지 않고 모든 데이터를 한 줄에 나오게 하려면, 다음 예와 같이 매개변수로 end=' ' 을 넣어 줌

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

- 띄어쓰기(space) 대신 콤마(,)를 넣고 싶은 경우

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

- 마지막 콤마를 출력하지 않는 경우

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

프로그래밍 연습 (1)

- 자연수 N을 입력 받은 다음, while문을 사용하여 1부터 N까지 자연수의 합을 계산하여 출력하는 프로그램 작성
- 동일한 프로그램을 for문을 사용하여 작성
- 프로그램의 실행 예

```
>>>  
N = 10  
합 : 55  
>>>  
>>>  
N = 20  
합 : 210  
>>>
```

% (나머지 연산자)

- % 연산자는 첫 번째 피연산자를 두 번째 피연산자로 나눈 나머지를 반환

```
>>> 5 % 3
2
>>> 10 % 2
0
>>> 7 % 3
1
```

- % 연산자를 수학에서는 modulo 연산자라고 부름
 - 순서도를 그릴 때는 modulo의 약자인 mod를 나머지 연산자로 사용함

== (관계 연산자)

- ==은 첫 번째 피연산자와 두 번째 피연산자가 같으면 True를 반환하고, 다르면 False를 반환
- !=은 첫 번째 피연산자와 두 번째 피연산자가 다르면 True를 반환하고, 같으면 False를 반환

```
>>> 5 % 3 == 2
True
>>> 5 % 3 == 3
False
>>> 5 % 3 != 2
False
>>> 5 % 3 != 3
True
```

프로그래밍 연습 (2)

- 자연수 N을 입력 받은 다음, N 이하의 홀수를 출력하는 프로그램 작성
- 홀수인지 구분할 때 나머지 연산자(%) 사용
- 동일한 프로그램을 while문과 for문을 사용하여 각각 작성
- 프로그램의 실행 예

```
>>>  
N = 20  
1 3 5 7 9 11 13 15 17 19  
>>>
```

프로그래밍 연습 (3)

- 자연수 N을 입력 받은 다음, 1을 입력하면 1부터 N 이하의 홀수, 2를 입력하면 2부터 N 이하의 짝수를 출력하는 프로그램 작성
- 홀수인지 짝수인지 구분할 때 나머지 연산자(%) 사용
- 동일한 프로그램을 while문과 for문을 사용하여 각각 작성
- 프로그램의 실행 예

```
>>>
N = 20
홀수/짝수 선택(1: 홀수, 2: 짝수) : 1
1 3 5 7 9 11 13 15 17 19
>>>
N = 15
홀수/짝수 선택(1: 홀수, 2: 짝수) : 2
2 4 6 8 10 12 14
>>>
N = 10
홀수/짝수 선택(1: 홀수, 2: 짝수) : 3
입력 오류
>>>
```


명령문의 들여쓰기 (1)



<규칙 1> 가장 큰 범위, 즉 가장 바깥의 코드 블록은 반드시 1열부터 시작해야 함

<규칙 2> 하나의 제어문에 속한 명령문들은 모두 같은 열에 위치해야 함

<규칙 3> 코드 블록의 끝은 들여쓰기가 끝나는 부분으로 간주함

명령문의 들여쓰기 (2)

- <규칙 1> 위반

```
>>> a = 0  
SyntaxError: unexpected indent
```

- <규칙 2> 위반

```
>>> i = 1  
>>> while i < 10:  
    print(i)  
    i += 1  
  
SyntaxError: unexpected indent
```

명령문의 들여쓰기 (3)

- 파이썬에서는 begin, end, {, } 등의 기호를 사용하지 않고, 들여쓰기만으로 코드 블록의 시작과 끝을 구분함
- 따라서 들여쓰기를 정확히 하지 않으면 목표한대로 프로그램이 실행되지 않을 수 있음
- <프로그램 1.3> 10 이하의 홀수 출력

```
i = 1
while i <= 10:
    if i % 2 == 1:
        print(i, end=' ')
    i += 1
```

- <프로그램 1.3>의 실행 결과

```
>>>
1 3 5 7 9
```

명령문의 들여쓰기 (4)

- <프로그램 1.4> 들여쓰기를 다르게 한 경우

```
i = 1
while i <= 10:
    if i % 2 == 1:
        print(i, end=' ')
        i += 1
```

- <프로그램 1.4>를 실행시키면 오류가 발생하지는 않지만, 다음과 같이 화면에 1이 출력된 후 프로그램이 멈춰 있음
 - 이런 경우에는 Ctrl-C를 눌러 실행을 중지시킴

```
>>>
1
```

명령문의 들여쓰기 (5)

- Ctrl-C를 눌러서 프로그램의 실행을 중지시켰을 때 나타나는 메시지

```
1 Traceback (most recent call last):  
  File "C:/Python34/ttt.py", line 2, in <module>  
    while i <= 10:  
KeyboardInterrupt
```

- 프로그램의 실행이 멈춘 이유
 - i의 값이 1인 경우에는 if문이 실행되어 i의 값이 2로 증가함
 - i의 값이 2가 된 다음부터는 if문이 실행되지 않기 때문에 i의 값이 증가되지 않아 while문을 벗어나지 못함
- 들여쓰기를 잘못하게 되면 프로그램을 실행시킬 때 오류가 나지는 않지만, 목표한 대로 프로그램이 실행되지 않을 수 있음

순서도 연산자 vs 파이썬 연산자

순서도의 연산자	파이썬 프로그램의 연산자
←	=
=	==
≠	!=
≥	>=
≤	<=
mod	%

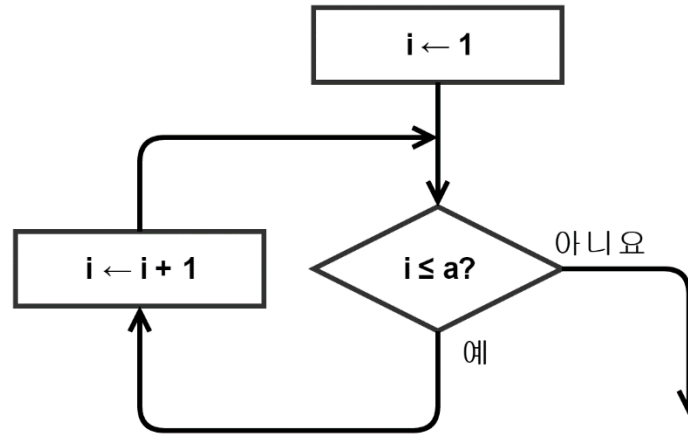
프로그램 작성

- 입력 받은 자연수의 모든 약수를 출력하는 프로그램 작성

```
>>>
자연수 입력: -1
-1 은(는) 자연수가 아닙니다.
>>>
자연수 입력: 12
12 의 모든 약수 : 1 2 3 4 6 12
>>>
자연수 입력: 24
24 의 모든 약수 : 1 2 3 4 6 8 12 24
>>>
```

반복문의 표현

- 순서도에서 반복문의 예



- while문

```
i = 1
while i <= a:
    i += 1
```

- for문

```
for i in range(1, a+1):
```


프로그래밍 과제

- 자연수가 아닌 수를 입력하면 자연수를 입력할 때까지 반복해서 데이터를 입력 받음
- 실행 예

```
>>>  
자연수 입력: -10  
-10 은(는) 자연수가 아닙니다.  
자연수 입력: 0  
0 은(는) 자연수가 아닙니다.  
자연수 입력: 24  
24 의 모든 약수 : 1 2 3 4 6 8 12 24  
>>>
```

1.3 최대공약수 구하기



프로그램 개요

- 두 개의 자연수 a 와 b 를 입력 받아서 a 와 b 의 최대공약수를 출력하는 프로그램을 작성

- 입력 : 자연수 a, b
- 출력 : a 와 b 의 최대공약수

- 최대공약수(greatest common divisor)
 - 두 개 이상의 수의 공약수 중 최대인 수
- 예) $a = 12, b = 18$
 - 12의 약수 : 1, 2, 3, 4, 6, 12
 - 18의 약수 : 1, 2, 3, 6, 9, 18
 - 12와 18의 공약수 : 1, 2, 3, 6
 - 12와 18의 최대공약수 : 6

파이썬 문법



논리 연산자 and

- 첫 번째 피연산자와 두 번째 피연산자가 모두 True일 경우에만 True를 반환하고, 나머지 경우에는 모두 False를 반환

```
>>> True and True
True
>>> True and False
False
>>> False and True
False
>>> False and False
False
```

- 현재 기온을 입력 받아 기온이 10도에서 30도 사이이면 “날씨가 좋다.”를 출력하는 프로그램을 작성할 때, 다음과 같이 논리 연산자 and를 사용

```
t = int(input('온도 : '))
if t <= 30 and t > 10: print('날씨가 좋다.')
```

논리 연산자 or

- 첫 번째 피연산자 또는 두 번째 피연산자 중 하나만 True여도 True를 반환하고, 둘 다 False일 경우에만 False를 반환

```
>>> True or True
True
>>> True or False
True
>>> False or True
True
>>> False or False
False
```

논리 연산자 not

- 피연산자가 True일 때는 False를 반환하고, False일 때는 True를 반환

```
>>> not True
False
>>> not False
True
```

논리 연산자의 진리표

x	y	x and y
True	True	True
True	False	False
False	True	False
False	False	False

x	y	x or y
True	True	True
True	False	True
False	True	True
False	False	False

x	not x
True	False
False	True

프로그래밍 연습

- 0이 아닌 두 정수 a와 b를 입력 받은 다음, 두 정수를 곱할 때 결과 값의 부호를 결정하는 프로그램 작성
- 예를 들어, a가 양수이고 b가 양수이면 결과 값이 양수가 되고, a가 양수이고, b가 음수이면 결과 값이 음수가 됨
- 프로그램의 실행 예

```
>>>  
a = 1  
b = -1  
음수  
>>>  
a = 20  
b = 30  
양수  
>>>
```

```
>>>  
a = -2  
b = 3  
음수  
>>>  
a = -1  
b = -2  
양수  
>>>
```

프로그램 작성

- 두 개의 자연수를 입력 받아 두 수의 최대공약수를 구하는 프로그램 작성

```
>>>
자연수 a 입력 : 12
자연수 b 입력 : 18
최대공약수 : 6
>>>
자연수 a 입력 : 9
자연수 b 입력 : 15
최대공약수 : 3
>>>
자연수 a 입력 : 6
자연수 b 입력 : 12
최대공약수 : 6
>>>
```

프로그래밍 과제 (1)

- 유클리드 호제법(Euclidean algorithm)을 사용하여 두 자연수의 최대공약수를 구하는 프로그램 작성
- 유클리드 호제법
 - 호제법이란 말은 두 수가 서로(互) 상대방 수를 나누어(除)서 결국 원하는 수를 얻는 알고리즘을 나타냄
 - 2개의 자연수(또는 정식) a , b 에 대해서 a 를 b 로 나눈 나머지를 r 이라 하면(단, $a > b$), a 와 b 의 최대공약수는 b 와 r 의 최대공약수와 같음
 - 이 성질에 따라, b 를 r 로 나눈 나머지 r' 를 구하고, 다시 r 을 r' 로 나눈 나머지를 구하는 과정을 반복하여 나머지가 0이 되었을 때 나누는 수가 a 와 b 의 최대공약수가 됨

프로그래밍 과제 (2)

- 유클리드 호제법의 예
 - 예를 들어, 720과 186의 최대공약수를 구하면,
 - 720은 186으로 나누어 떨어지지 않기 때문에, 720을 186으로 나눈 나머지를 구하면, 나머지는 162이 됨
 - 186은 162로 나누어 떨어지지 않기 때문에, 186을 162로 나눈 나머지를 구하면, 나머지는 24가 됨
 - 162는 24로 나누어 떨어지지 않기 때문에, 162를 24로 나눈 나머지를 구하면, 나머지는 18이 됨
 - 24는 18로 나누어 떨어지지 않기 때문에, 24를 18로 나눈 나머지를 구하면, 나머지는 6이 됨
 - 18은 6으로 나누어 떨어짐
 - 따라서, 최대공약수는 6이 됨

1.4 최소공배수 구하기



프로그램 개요

- 두 개의 자연수 a 와 b 를 입력 받아서 a 와 b 의 최소공배수를 출력하는 프로그램을 작성

- 입력 : 자연수 a, b
- 출력 : a 와 b 의 최소공배수

- 최소공배수(least common multiple)
 - 두 개 이상의 수의 공배수 중 최소인 수
- 예) $a = 12, b = 18$
 - 12의 배수 : 12, 24, 36, 48, ...
 - 18의 배수 : 18, 36, 54, 72, ...
 - 12와 18의 공배수 : 36, 72, ...
 - 12와 18의 최소공배수 : 36

프로그램 작성

- 두 개의 자연수를 입력 받아 두 수의 최소공배수를 구하는 프로그램 작성

```
>>>
자연수 a 입력 : 12
자연수 b 입력 : 18
최소공배수 : 36
>>>
자연수 a 입력 : 10
자연수 b 입력 : 10
최소공배수 : 10
>>>
자연수 a 입력 : 10
자연수 b 입력 : 20
최소공배수 : 20
>>>
```

프로그래밍 과제

- 세 개의 자연수를 입력 받아 세 수의 최소공배수를 구하는 프로그램 작성

```
>>>  
자연수 a 입력 : 2  
자연수 b 입력 : 3  
자연수 c 입력 : 4  
최소공배수 : 12
```

```
>>>  
자연수 a 입력 : 10  
자연수 b 입력 : 20  
자연수 c 입력 : 40  
최소공배수 : 40
```

```
>>>  
자연수 a 입력 : 12  
자연수 b 입력 : 18  
자연수 c 입력 : 24  
최소공배수 : 72  
>>>
```


1.5 완전수 여부 판별



프로그램 개요

- 자연수 a 를 입력 받아서 a 가 완전수인지 여부를 판별하는 프로그램을 작성

- 입력 : 자연수 a
- 출력 : a 의 완전수 여부

- 완전수(perfect number)
 - 자신을 제외한 모든 약수들의 합이 자신과 같은 수
- 예) 6은 완전수
 - 6의 모든 약수 : 1, 2, 3, 6
 - 자신을 제외한 약수 1, 2, 3의 합 : 6

프로그램 작성

- 입력 받은 자연수가 완전수인지 여부를 출력하는 프로그램 작성

```
>>>  
자연수 입력 : 6  
6 은(는) 완전수입니다.  
>>>  
자연수 입력 : 12  
12 은(는) 완전수가 아닙니다.  
>>>
```

프로그래밍 과제

- 2보다 큰 자연수 N을 입력 받아 2부터 N까지의 자연수 중에서 완전수를 출력하는 프로그램 작성

```
>>>  
N 입력 : 1  
N은 2보다 큰 자연수여야 합니다.  
N 입력 : 10000  
6 28 496 8128  
>>>
```

1.6 소수 여부 판별



프로그램 개요

- 자연수 a 를 입력 받아서 a 가 소수인지 여부를 판별하는 프로그램을 작성

- 입력 : 자연수 a
- 출력 : a 의 소수 여부

- 소수(prime number)
 - 1보다 큰 자연수 중에서 1과 자기 자신만으로 나누어떨어지는 수
 - 1보다 큰 자연수 중에서 1과 자기 자신 2개만을 약수로 가지는 수
- 예) 2, 3, 5, 7, 11, 13, 17, 19, 23, 29, 31,...

파이썬 문법



자료형 (1)

- 자료형(data type)이란?
 - 어떤 변수에 저장되는 값의 종류와 범위를 지정
 - a라는 변수에 정수 3을 저장하려고 하면 변수 a의 자료형을 정수(int)로 지정해야 하고, 실수 3.1을 저장하려고 하면 변수 a의 자료형을 실수(float)로 지정해야 함
 - C 언어의 경우에는 같은 숫자라고 해도 int, short, unsigned int, unsigned short, float, double, ... 등 메모리(memory)를 얼마나 차지하는지 또는 숫자를 어떻게 표현하는지에 따라 자료형이 세분화되어 있음
 - 따라서 이렇게 세분화되어 있는 자료형을 정확하게 사용하면 프로그램의 성능을 향상시킬 수 있지만, 잘못 사용하면 성능을 저하시킬 수도 있고, 아니면 알 수 없는 오작동이나 오류가 발생할 수도 있음

자료형 (2)

- 파이썬의 자료형
 - 매우 단순하고 직관적
 - 파이썬을 사용해서 프로그래밍을 할 때에는 변수의 자료형을 무엇으로 지정할지 고민할 필요 없이 그냥 사용하면 됨
 - 동적 타이핑(dynamic typing) 언어 : 파이썬처럼 변수의 자료형을 미리 지정하지 않는 프로그래밍 언어
 - 파이썬에서는 변수에 값을 저장하면 스스로 알아서 자료형을 알아낸 다음 그에 적합한 객체를 만들어줌
- type() 함수 : 변수의 자료형을 반환

```
>>> a = 3
>>> type(a)
<class 'int'>
>>> a = 3.1
>>> type(a)
<class 'float'>
>>> a = 'abc'
>>> type(a)
<class 'str'>
```

자료형 (3)

- 파이썬에서는 자료형에 대한 변환(casting) 없이도 서로 다른 자료형을 가지고 있는 변수들끼리 자유롭게 계산을 할 수 있음

```
>>> a = 3
>>> b = 2.4
>>> a + b
5.4
>>> a * b
7.199999999999999
>>> a = 3
>>> b = 2+3j
>>> a + b
(5+3j)
>>> a * b
(6+9j)
```

```
>>> a = 2.4
>>> b = 2+4j
>>> a + b
(4.4+4j)
>>> a * b
(4.8+9.6j)
>>> a = 1+2j
>>> b = 2+4j
>>> a + b
(3+6j)
>>> a * b
(-6+8j)
```

부울 자료형 (1)

- 부울 자료형(bool data type)
 - 변수에 참(True)과 거짓(False) 값만을 저장하는데 사용됨
 - 파이썬에서 부울 자료형은 정수 자료형의 부분형(subtype)
 - 어떤 변수에 True를 저장하면 실제로는 정수 1이 저장되고, False를 저장하면 실제로는 정수 0이 저장됨

```
>>> a = True
>>> a
True
>>> type(a)
<class 'bool'>
>>> b = False
>>> a + b
1
>>> a * b
0
```

부울 자료형 (2)

- 조건문에서 부울 자료형을 가지는 변수를 사용하는 방법
 - if문에서 "if a:"라고 쓰면, a가 True 값을 가지고 있을 때는 if 뒤에 있는 명령문이 실행되고, a가 False 값을 가지고 있을 경우에는 else 뒤에 있는 명령문이 실행됨

```
>>> a = True
>>> b = False
>>> if a: print('참')
else: print('거짓')
참
>>> a = b
>>> if a: print('참')
else: print('거짓')
거짓
```

함수 (1)

- 함수(function)
 - 동일한 작업을 프로그램의 여러 곳에서 여러 번 수행하고 싶을 때 사용
- 예) N 이하의 자연수 중에서 홀수만 출력하는 프로그램

```
N = int(input('자연수 N 입력 : '))
for i in range(1, N+1):
    if i % 2 == 1:
        print(i, end=' ')
```

- N의 값으로 10을 입력했을 경우 실행 결과

```
>>>
자연수 N 입력 : 10
1 3 5 7 9
>>>
```

함수 (2)

- 함수의 구조

```
def 함수명(매개변수):  
    명령문  
    return 변수명
```

함수 (3)

- isOdd() 함수를 사용한 프로그램

```
def isOdd(num):  
    if num % 2 == 1: return True  
    else: return False
```

```
N = int(input('자연수 N 입력 : '))  
for i in range(1, N+1):  
    if isOdd(i): print(i, end=' ')
```

- 함수의 수행 과정
 - " if isOdd(i): " 문에서 함수 isOdd(i)를 호출하는데, 이 함수의 매개변수가 i이므로 1부터 N까지 i의 값이 차례대로 isOdd(num) 함수로 전달
 - isOdd(num) 함수에 있는 매개변수 num은 i의 값을 전달받은 다음, num을 2로 나눈 나머지가 1이면 True를 반환하고, 그렇지 않으면 False를 반환

함수 (4)

- x의 y 제곱을 계산해 주는 함수

```
def power(x, y):  
    res = x  
    for i in range(y-1):  
        res *= x  
    return res
```

```
a = int(input('a = '))  
b = int(input('b = '))  
result = power(a, b)
```

- 함수의 수행 과정
 - res에 x를 저장하고 나서 res에 x를 곱한 결과를 res에 저장하는 단계를 y-1번 반복
 - 예를 들어, x에 2를 넣고, y에 5를 넣은 경우 처음 res는 2가 되고, res에 2를 곱하는 단계를 4번 더 반복하면 res는 32가 됨

프로그래밍 연습 (1)

- 매개변수로 주어진 세 정수 x, y, z 의 최대값을 찾아서 반환해 주는 함수 $\text{max3}(x, y, z)$ 를 사용하여 입력된 세 정수 a, b, c 의 최대값을 구하는 프로그램 작성
- 프로그램의 일부

```
def max3(x, y, z):  
  
a = int(input('a = '))  
b = int(input('b = '))  
c = int(input('c = '))  
maxval = max3(a, b, c)  
print('최대값 : ', maxval)
```

프로그래밍 연습 (2)

- 매개변수로 주어진 n 이 m 의 약수인지 여부를 판별하는 함수 `isDivisor(m, n)`을 사용하여 자연수 a 의 모든 약수를 구하는 프로그램 작성
- 프로그램의 일부

```
def isDivisor(m, n):  
  
a = int(input('a = '))  
for i in range(1, a+1):  
    if isDivisor(a, i):  
        print(i, end=' ')
```

프로그래밍 연습 (3)

- 매개변수로 주어진 m 과 n 의 최대공약수를 반환하는 함수 $\text{gcd}(m, n)$ 을 사용하여 자연수 a 와 b 의 최대공약수를 구하는 프로그램 작성
- 프로그램의 일부

```
def gcd(m, n):  
  
a = int(input('a = '))  
b = int(input('b = '))  
print('최대공약수 : ', gcd(a, b))
```

프로그래밍 연습 (4)

- 매개변수로 주어진 m과 n의 최소공배수를 반환하는 함수 lcm(m, n)을 사용하여 자연수 a와 b의 최소공배수를 구하는 프로그램 작성
- 프로그램의 일부

```
def lcm(m, n):
```

```
    a = int(input('a = '))
```

```
    b = int(input('b = '))
```

```
    print('최소공배수 : ', lcm(a, b))
```

프로그래밍 연습 (5)

- 매개변수로 주어진 m이 완전수인지 판별하는 함수 isPerfect(m)을 사용하여, 2부터 N까지의 자연수 중에서 완전수를 출력하는 프로그램 작성
- 프로그램의 일부

```
def isPerfect(m):  
  
N = int(input('N = '))  
for i in range(1, N+1):  
    if isPerfect(i):  
        print(i, end=' ')
```

지역 변수와 전역 변수 (1)

- 지역 변수(local variable)
 - 함수 내에서만 사용할 수 있는 변수
- 전역 변수(global variable)
 - 프로그램 전체적으로 사용할 수 있는 변수

지역 변수와 전역 변수 (2)

- 두 정수의 합을 출력하는 프로그램
 - add(a, b) 함수에 있는 변수 c는 지역 변수이므로 함수 외부로 변경된 값을 전달하지 못함

```
def add(a, b):  
    c = a + b
```

```
a = int(input('정수 입력 : '))  
b = int(input('정수 입력 : '))  
c = 0  
add(a, b)  
print('c = ', c)
```

- 실행 결과

```
>>>  
정수 입력 : 1  
정수 입력 : 2  
c = 0
```

지역 변수와 전역 변수 (3)

- 전역 변수 사용
 - 변수 c의 변경된 값이 함수 외부로 전달

```
def add(a, b):  
    global c  
    c = a + b  
  
a = int(input('정수 입력 : '))  
b = int(input('정수 입력 : '))  
c = 0  
add(a, b)  
print('c = ', c)
```

- 실행 결과

```
>>>  
정수 입력 : 1  
정수 입력 : 2  
c = 3
```


프로그래밍 연습 (6)

- 함수 `sum_avg(x, y, z)`를 사용하여 입력된 세 정수 `a`, `b`, `c`의 합과 평균을 구하는 프로그램 작성
- 반환해야 하는 값이 2개 이상일 경우 전역 변수를 사용하거나 튜플 자료형을 사용함
- 여기서는 전역 변수를 사용하여 프로그램 작성
- 프로그램의 실행 예

```
>>>  
a = 3  
b = 4  
c = 5  
합 : 12  
평균 : 4.0  
>>>
```

프로그램 작성

- 입력 받은 자연수가 소수인지 여부를 출력하는 프로그램 작성

```
>>>
2 이상의 자연수 입력 : 49
49 은(는) 소수가 아닙니다.
>>>
2 이상의 자연수 입력 : 13
13 은(는) 소수입니다.
>>>
```

프로그래밍 과제

- 2 이상의 자연수 N을 입력 받아 2부터 N 사이의 자연수 중에서 소수만 출력하는 프로그램 작성
- 실행 예

```
>>>
2 이상의 자연수 입력 : 1
1 은(는) 2 이상의 자연수가 아닙니다.
2 이상의 자연수 입력 : 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
>>>
```

1.7 에라토스테네스의 체



프로그램 개요 (1)

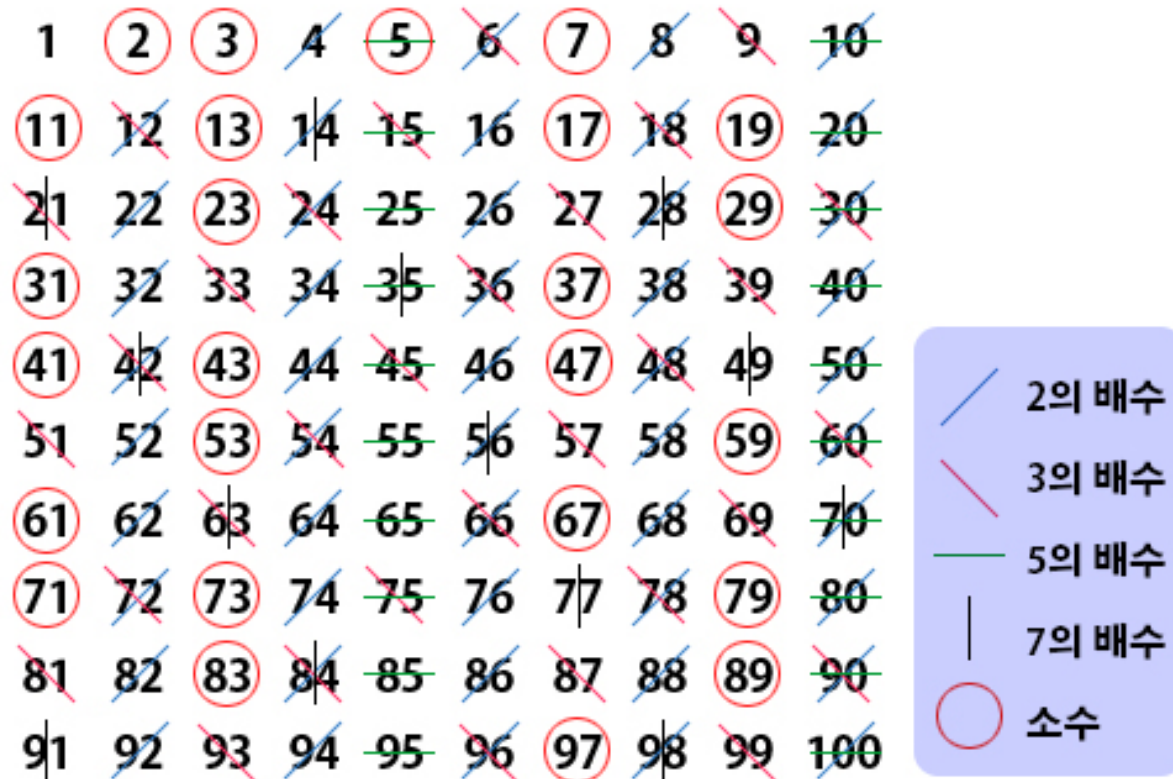
- 에라토스테네스의 체 방법을 사용하여 N 이하의 자연수 중에서 소수만 출력하는 프로그램을 작성

- 입력 : 2 이상의 자연수 N
- 출력 : N 이하의 소수

- 에라토스테네스(Eratosthenes)
 - 그리스의 시인이자 천문학자이며, 지리학자 겸 수학자
 - 경도가 비슷한 두 도시 시에네와 알렉산드리아에 해시계를 두고 하룻날 정오에 해시계의 바늘이 만드는 그림자의 각도 차이를 이용하여 지구의 둘레를 계산한 사람으로 유명함

프로그램 개요 (2)

- 에라토스테네스의 체(Sieve of Eratosthenes)



파이썬 문법



리스트 자료형 (1)

- 리스트 자료형(list data type)
 - 순서대로 데이터를 저장하고 싶을 경우에 사용
 - C나 Java와 같은 프로그래밍 언어에서는 동일한 자료형의 데이터를 순서대로 저장하기 위해 배열(array)을 사용함
 - 파이썬에서 배열과 가장 비슷한 역할을 하는 것이 리스트 자료형임
- 리스트 자료형의 예

```
>>> a = [10, 20, 30]
>>> a
[10, 20, 30]
>>> type(a)
<class 'list'>
```

- 리스트는 다양한 자료형의 데이터를 원소로 가질 수 있음

```
>>> b = [10, 'abc', True, 3.5]
>>> b
[10, 'abc', True, 3.5]
```


리스트 자료형 (2)

- 인덱싱(indexing)
 - 인덱스를 사용하여 리스트의 원소를 접근하는 방법
 - 리스트의 인덱스는 0부터 시작하여 차례대로 1씩 증가
- 인덱싱의 예

```
>>> a[0]  
10  
>>> a[1]  
20  
>>> a[2]  
30
```

- 범위를 벗어나는 인덱스를 사용하면 오류가 발생함

```
>>> a[3]  
Traceback (most recent call last):  
File "<pyshell#33>", line 1, in <module>  
a[3]  
IndexError: list index out of range
```

리스트 자료형 (3)

- 슬라이싱(slicing)
 - 리스트 원소들에 대해 범위를 지정하는 방법

```
>>> b[0:2]
[10, 'abc']
>>> b[1:3]
['abc', True]
>>> b[1:4]
['abc', True, 3.5]
```

리스트 자료형 (4)

- 중첩 리스트(nested list)
 - 리스트 내에 중첩해서 리스트를 가질 수 있음

```
>>> c = [[1, 2, 3], [4, 5], [6], [7, 8, 9, 10]]
>>> c
[[1, 2, 3], [4, 5], [6], [7, 8, 9, 10]]
>>> c[0]
[1, 2, 3]
>>> c[3]
[7, 8, 9, 10]
>>>
>>> c[0][0]
1
>>> c[3][2]
9
```

리스트 자료형 (5)

- 중첩 리스트 출력

```
>>> for i in range(len(c)): print(c[i])
```

```
[1, 2, 3]
```

```
[4, 5]
```

```
[6]
```

```
[7, 8, 9, 10]
```

- 중첩 리스트의 모든 원소를 출력하는 프로그램

```
for i in range(len(c)):
    for j in range(len(c[i])):
        print(c[i][j])
```

리스트 자료형 (6)

- 3차원 리스트의 예

```
>>> d = [[[1, 2], [3, 4, 5]], [[6, 7, 8], [9, 10]]]
>>> d
[[[1, 2], [3, 4, 5]], [[6, 7, 8], [9, 10]]]
>>> d[0]
[[1, 2], [3, 4, 5]]
>>> d[0][0]
[1, 2]
>>> d[0][0][0]
1
>>> d[0][1][1]
4
>>> d[1][1][1]
10
```

리스트 자료형 (7)

- 메소드(method)
 - 객체 지향 프로그래밍 언어에서 메시지에 따라 실행시키는 프로시저
- 리스트의 내장 메소드 (1)

메소드	동작
append(x)	x를 리스트의 마지막 원소로 추가
insert(i, x)	x를 i번째 위치로 삽입
remove(x)	x와 같은 값을 가지고 있는 원소 중 첫 번째 원소를 삭제
index(x)	x와 같은 값을 가지고 있는 원소 중 첫 번째 원소의 인덱스를 반환
count(x)	x와 같은 값을 가지고 있는 원소의 개수를 반환

리스트 자료형 (8)

- 리스트의 내장 메소드 (2)

메소드	동작
extend(list)	원 리스트와 list를 병합하여 확장
clear()	모든 원소를 삭제
pop() pop(i)	마지막 원소를 삭제하고 반환 i번째 원소를 삭제하고 반환
sort()	오름차순으로 정렬
reverse()	리스트의 원소를 역순으로 배열
copy()	얕은(shallow) 복사본을 반환

리스트 자료형 (9)

- 내장 메소드를 사용한 리스트 연산 (1)

```
>>> a = [1, 2, 2, 3, 3, 3]
>>> a
[1, 2, 2, 3, 3, 3]
>>> print(a.count(1), a.count(2), a.count(3), a.count(4))
1 2 3 0
>>> a.append(4)
>>> a
[1, 2, 2, 3, 3, 3, 4]
>>> a.insert(2, 5)
>>> a
[1, 2, 5, 2, 3, 3, 3, 4]
>>> a.index(5)
2
>>> a.remove(3)
>>> a
[1, 2, 5, 2, 3, 3, 4]
```


리스트 자료형 (10)

- 내장 메소드를 사용한 리스트 연산 (2)

```
>>> a.reverse()
>>> a
[4, 3, 3, 2, 5, 2, 1]
>>> a.sort()
>>> a
[1, 2, 2, 3, 3, 4, 5]
>>> a.pop()
5
>>> a
[1, 2, 2, 3, 3, 4]
>>> b = a.copy()
>>> b
[1, 2, 2, 3, 3, 4]
>>> b.clear()
>>> b
[]
```

리스트 자료형 (11)

- append() 메소드와 extend() 메소드

```
>>> a = [1, 2, 3, 4]
>>> a
[1, 2, 3, 4]
>>> a.append([5, 6])
>>> a
[1, 2, 3, 4, [5, 6]]
>>> a = [1, 2, 3, 4]
>>> a.extend([5, 6])
>>> a
[1, 2, 3, 4, 5, 6]
```

프로그래밍 연습 (1)

- count()와 append() 메소드를 사용하여 리스트의 데이터를 중복 없이 입력하는 프로그램 작성
- 프로그램의 실행 예

```
>>>
정수 입력(종료시는 999) : 1
정수 입력(종료시는 999) : 2
정수 입력(종료시는 999) : 2
정수 입력(종료시는 999) : 3
정수 입력(종료시는 999) : 3
정수 입력(종료시는 999) : 3
정수 입력(종료시는 999) : 999
[1, 2, 3]
>>>
```

range() 함수 (1)

- range() 함수

range(시작범위, 끝범위, 증가치)

- 시작범위와 증가치는 생략할 수 있는데, 시작범위를 생략하면 시작범위는 0이 되고, 증가치를 생략하면 증가치는 1이 됨

- range() 함수를 사용하는 방법

구 분	설 명
range(n)	0부터 n-1까지 range 자료형 반환
range(m, n)	m부터 n-1까지 range 자료형 반환
range(m, n, s)	m부터 시작하여 s만큼씩 증가시키면서 n-1까지 range 자료형 반환

range() 함수 (2)

- range() 함수 사용 예

```
>>> list(range(5))  
[0, 1, 2, 3, 4]  
>>> list(range(5, 10))  
[5, 6, 7, 8, 9]  
>>> list(range(1, 10, 2))  
[1, 3, 5, 7, 9]
```

- range 자료형 사용 예

```
>>> a = range(10)  
>>> a  
range(0, 10)  
>>> type(a)  
<class 'range'>
```

프로그래밍 연습 (2)

- 자연수 N과 m을 입력 받은 다음, N 이하의 m의 배수를 출력하는 프로그램 작성
- for문에서 사용하는 range() 함수의 매개변수를 통해 증가치를 설정
- 프로그램의 실행 예

```
>>>  
N = 20  
m = 3  
3 6 9 12 15 18  
>>>  
N = 30  
m = 4  
4 8 12 16 20
```

len() 함수

- len() 함수는 문자열이나 리스트의 길이를 반환

```
>>> s = 'string'
>>> s
'string'
>>> len(s)
6
>>> s = '문자열'
>>> s
'문자열'
>>> len(s)
3
>>> a = [1, 2, 3, 4, 5]
>>> a
[1, 2, 3, 4, 5]
>>> len(a)
5
```

list() 함수

- 1부터 10까지의 정수를 리스트 a[]의 원소로 차례대로 저장하고 싶은 경우

```
>>> a = list(range(1, 11))  
>>> a  
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]  
>>>
```

- list() 함수 동일한 동작을 수행하는 while문

```
a = []  
i = 0  
while i <= N:  
    a.append(i)  
    i += 1
```


프로그램의 실행시간 측정 (1)

- 프로그램의 실행시간을 측정하는 방법

```
import time
start = time.time()
# 파이썬 코드
end = time.time() - start
print('실행시간 :', end)
```

프로그램의 실행시간 측정 (2)

- time 모듈의 메소드 (1)

- time() 메소드 : 1970년 1월 1일 00시 00분 00초부터 시작하여 현재까지 몇 초가 흘렀는지를 나타냄

```
>>> import time
>>> time.time()
1450057281.323308
```

- asctime() 메소드 : 현재까지 흐른 초가 아니라 연월일시를 표시

```
>>> time.asctime()
'Sun Dec 13 19:57:20 2015'
```

- sleep() 메소드 : 프로그램을 실행시킬 때 중간에 잠깐 쉬고 싶은 경우나 프로그램을 천천히 실행시키고 싶은 경우에 사용

```
>>> for i in range(10):
    print(i)
    time.sleep(1)
```

프로그램의 실행시간 측정 (3)

- time 모듈의 메소드 (2)
 - localtime() 메소드 : 날짜와 시간을 객체로 반환하기 때문에 다음 예와 같이 연월일시를 각각 처리할 수 있음

```
>>> time.localtime()
time.struct_time(tm_year=2015, tm_mon=12, tm_mday=13,
tm_hour=20, tm_min=0, tm_sec=56, tm_wday=6, tm_yday=347,
tm_isdst=0)
>>> t = time.localtime()
>>> year = t[0]
>>> month = t[1]
>>> print(year, '년', month, '월')
2015 년 12 월
```

프로그램의 실행시간 측정 (4)

- 프로그램의 실행시간 측정 예
 - 1부터 N까지 합을 구하는 프로그램

```
import time
N = int(input('자연수 N 입력 : '))
s = 0
start = time.time()
for i in range(1, N+1):
    s += i
end = time.time() - start
print('합계 : ', s)
print('실행시간 : ', end)
```

프로그래밍 연습 (3)

- 1부터 N^2 까지 합을 구하는 프로그램의 실행시간 측정
- 프로그램의 실행 예

```
>>>
자연수 N 입력 : 1000
합계 : 500000500000
실행시간 : 0.13315606117248535
>>>
자연수 N 입력 : 2000
합계 : 8000002000000
실행시간 : 0.5829799175262451
>>>
```

프로그램 작성 (1)

- 에라토스테네스의 체 방법을 사용하여, 2 이상의 자연수 N을 입력 받아 2부터 N 사이의 자연수 중에서 소수만 출력하는 프로그램 작성

```
>>>
2 이상의 자연수 입력 : 0
0 은(는) 2 이상의 자연수가 아닙니다.
2 이상의 자연수 입력 : 100
2 3 5 7 11 13 17 19 23 29 31 37 41 43 47 53 59 61 67 71 73 79 83
89 97
>>>
```

프로그램 작성 (2)

- i 의 값이 2에서 5까지 변할 때 리스트에 저장된 값이 변화하는 모습 ($N = 50$)

- 최초 리스트

```
[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34, 35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50]
```

- $i = 2$: 모든 2의 배수가 0이 됨

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 9, 0, 11, 0, 13, 0, 15, 0, 17, 0, 19, 0, 21, 0, 23, 0, 25, 0, 27, 0, 29, 0, 31, 0, 33, 0, 35, 0, 37, 0, 39, 0, 41, 0, 43, 0, 45, 0, 47, 0, 49, 0]
```

프로그램 작성 (3)

- $i = 3$: 모든 3의 배수가 0이 됨

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0, 0, 0, 23, 0, 25, 0, 0, 0, 29, 0, 31, 0, 0, 0, 35, 0, 37, 0, 0, 0, 41, 0, 43, 0, 0, 0, 47, 0, 49, 0]
```

- $i = 4$: 모든 4의 배수가 0이 됨

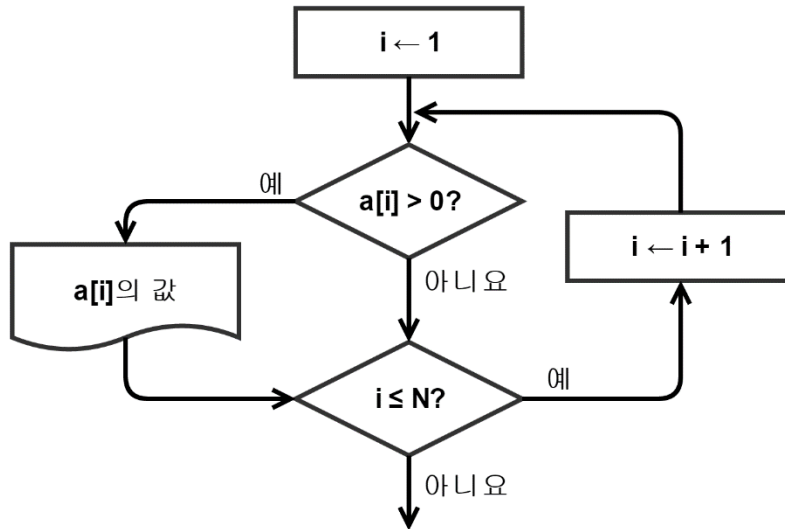
```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0, 0, 0, 23, 0, 25, 0, 0, 0, 29, 0, 31, 0, 0, 0, 35, 0, 37, 0, 0, 0, 41, 0, 43, 0, 0, 0, 47, 0, 49, 0]
```

- $i = 5$: 모든 5의 배수가 0이 됨

```
[0, 0, 2, 3, 0, 5, 0, 7, 0, 0, 0, 11, 0, 13, 0, 0, 0, 17, 0, 19, 0, 0, 0, 23, 0, 0, 0, 0, 0, 29, 0, 31, 0, 0, 0, 0, 0, 37, 0, 0, 0, 41, 0, 43, 0, 0, 0, 47, 0, 49, 0]
```


프로그램 작성 (4)

- 0이 아닌 $a[i]$ 의 값을 출력하는 부분
 - 순서도와 소스 코드



```
i = 1
while i <= N:
    if a[i] > 0: print(a[i], end=' ')
    i += 1
```

```
for i in range(1, N+1):
    if a[i]: print(a[i], end=' ')
```

프로그래밍 과제 (1)

- N에 100,000 이상의 큰 값을 입력했을 때, 에라토스테네스의 체 프로그램의 실행시간을 측정하는 프로그램 작성

```
>>>
2 이상의 자연수 입력 : 500000
실행시간 : 2.828158140182495
41538 개의 소수 발견
>>>
2 이상의 자연수 입력 : 1000000
실행시간 : 5.965723037719727
78498 개의 소수 발견
>>>
```

프로그래밍 과제 (2)

- N에 큰 값을 입력했을 경우 프로그램의 실행시간을 줄일 수 있는 방법
 - 프로그램의 실행시간을 줄이기 위해서는 프로그램에서 반복해서 수행되는 부분 중 불필요하게 중복해서 수행되는 부분을 없애야 함
 - 예를 들어, 이 프로그램의 경우에는 리스트 a의 원소 중에서 2의 배수를 모두 0으로 변경하였으면, 4의 배수, 6의 배수, 8의 배수 등이 모두 0으로 변경되었기 때문에, 4의 배수, 6의 배수, 8의 배수 등에 대해서는 0으로 만들 필요가 없음
 - 또한 3의 배수를 모두 0으로 변경하였으면, 6의 배수, 9의 배수 등에 대해서도 중복해서 수행할 필요가 없음

```
>>>
2 이상의 자연수 입력 : 500000
실행시간 : 0.9218831062316895
41538 개의 소수 발견
>>>
2 이상의 자연수 입력 : 1000000
실행시간 : 1.9218947887420654
78498 개의 소수 발견
>>>
```

continue-break (1)

- 반복문에서 벗어나는 방법을 제공

```
a = list(range(1, 11))
print(a)
for i in a:
    if i == 8:
        break
    if i % 2 == 0:
        continue
    print(i, end=' ')
print()
print(i)
```

continue-break (2)

- 에라토스테네스의 체

```
N = int(input('N = '))
a = list(range(N+1))
a[1] = 0
for i in range(2, int(N/2)+1):
    for j in range(i, int(N/2)+1):
        if i*j <= N and a[i*j] == 0:
            continue
        if i*j <= N:
            a[i*j] = 0
        else:
            break
for i in range(N+1):
    if a[i]:
        print(a[i], end=' ')
```