

따라하며 배우는

파이썬과 데이터 과학



3장 연산자로 계산을 해
보자

이장에서 배울 것들

- 연산이 무엇인지 이해해 보아요.
- 더하기, 빼기, 곱하기, 나누기 등의 산술 연산자에 대하여 학습해 보아요.
- 나머지, 지수 연산자에 대하여 익혀 보아요.
- 할당 연산자와 복합 연산자를 활용할 수 있게 될 것이에요.
- 프로그램을 이해하는데 꼭 필요한 주석의 개념을 이해해 보아요.
- 우선순위의 개념을 익히고 활용해 보아요.
- 산술 연산과 관련된 응용 프로그램을 작성할 수 있어요.

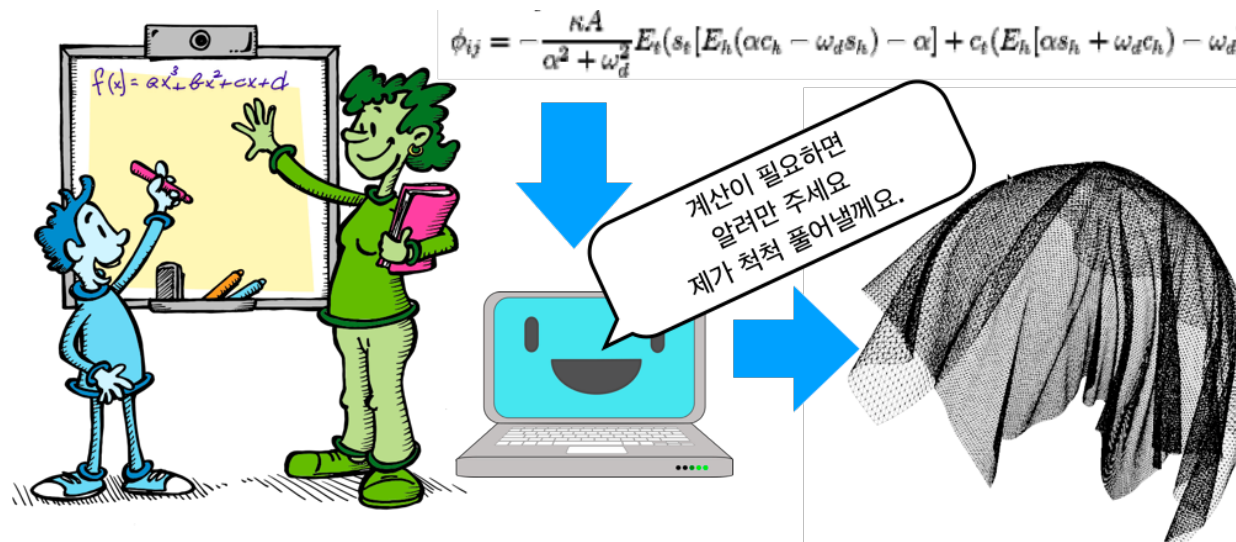
3.1 수식은 어디에나 있다

- 우리가 즐겨보는 영화의 컴퓨터 그래픽 장면들이 컴퓨터의 계산 기능을 통하여 이루어진다는 것은 아주 흥미롭다.
- 예를 들어서 다음 그림처럼 천의 움직임을 컴퓨터로 합성해 내는 일은 천의 동작을 수학적으로 모델링하는 수식 없이는 불가능하다.



3.1 수식은 어디에나 있다

- 많은 사람들은 수학을 두려워한다. 그러나 안심해도 좋다. 이번 장에서는 우리가 직접 계산할 필요는 없다. 수식만 작성하여 컴퓨터로 넘기면 컴퓨터가 알아서 수식을 계산한다. 컴퓨터는 수학을 두려워하지 않을 뿐 아니라 매우 빠른 속도로 정확하게 계산을 하여 결과를 알려줄 것이다.



3.1 수식은 어디에나 있다

- 컴퓨터가 수식 계산을 담당하는데 왜 우리가 수식에 대해 알아야 할까?
- 올바르게 수식을 이해하고 잘 작성하여 계산의 효율도 높이고 코드도 간략하게 만들 수 있다. 이런 일을 위해 연산자가 적용되는 순서도 알아야 하며, 사칙 연산과 달리 평소 잘 사용하지 않던 연산자도 익히게 될 것이다.



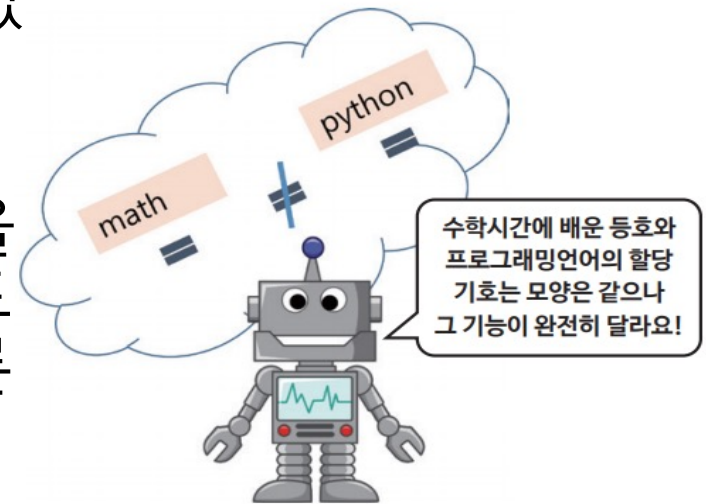
잠깐 - 수학의 수식과 프로그래밍의 수식은 다른 개념이다

수학의 수식 **mathematical expression**은 수학적 표기법과 기호를 이용하여 수학적 개념을 표현하는 것이다. 수식을 통해 우리는 수식 내의 기호들이 표현하는 수학적 대상들의 관계를 파악할 수 있다.

컴퓨터 과학분야의 프로그래밍에서 사용하는 **수식** **expression**은 수학적 관계를 나타내는 것이 아니라 하나의 값으로 **평가** **evaluation**될 수 있는 표현을 의미한다. 예를 들어 $7 * 7 + 10$ 이라는 표현은 50이라는 값으로 평가될 수 있는 수식이다.

3.2 할당 연산자를 통해 변수에 값을 넣자

- 우리는 이미 연산자 하나를 사용하고 있었다. 바로 할당 연산자 `assignment operator`이다. 변수에 값을 할당할 때 사용하였던 `=` 기호가 바로 할당 연산자인데 대입 연산자라고 부르기도 한다.
- 이것은 "같다"라는 의미가 아니고, 변수에 값을 저장하는 연산자이다. `=` 연산자의 왼쪽은 반드시 값을 저장할 수 있는 변수이어야 하고, 오른쪽은 값을 표현하는 숫자나 변수, 혹은 수식 `expression`이 와야 한다.

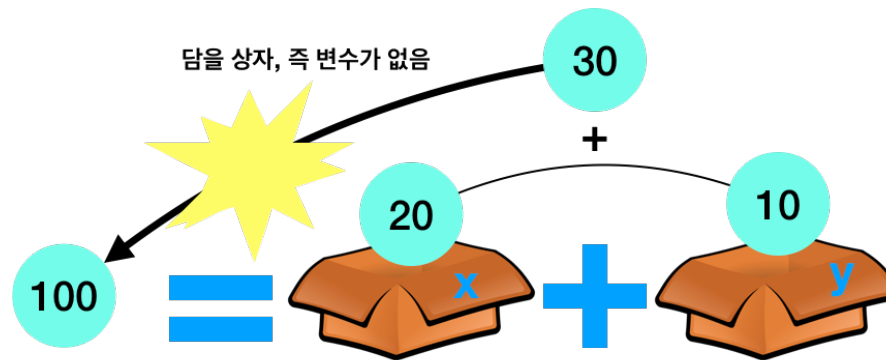


```
>>> x = 100 + 200    # x에 100 + 200의 결과를 할당
>>> x
300
```

3.2 할당 연산자를 통해 변수에 값을 넣자

- 등호의 왼쪽이 변수가 아니면 문제가 발생한다. 다음과 같은 코드는 잘못된 문법 오류를 발생시킨다

```
>>> x = 20
>>> y = 10
>>> 100 = x + y    # 등호의 왼쪽에는 반드시 변수이름이 와야한다
File "<stdin>", line 1
SyntaxError: cannot assign to literal
```



3.2 할당 연산자를 통해 변수에 값을 넣자

- 다음과 같이 여러 개의 변수에 동일한 값을 저장하는 것도 가능하다. $x = y = 100$ 과 같이 할 경우 x, y 의 값이 모두 100으로 할당되며 이를 **다중 할당문** *multiple assignment*이라 한다.
- 그리고 $n1, n2 = 100, 200$ 과 같이 여러 변수에 한꺼번에 여러 값을 할당할 수도 있는데 이를 **동시 할당문** *simultaneous assignment*이라 한다.

```
>>> x = y = 100          # 여러 변수에 동일한 값을 할당하는 다중 할당문
>>> x, y
(100, 100)
>>> n1, n2 = 100, 200    # 여러 변수에 한꺼번에 여러 값을 할당하는 동시 할당문
>>> n1, n2
(100, 200)
```

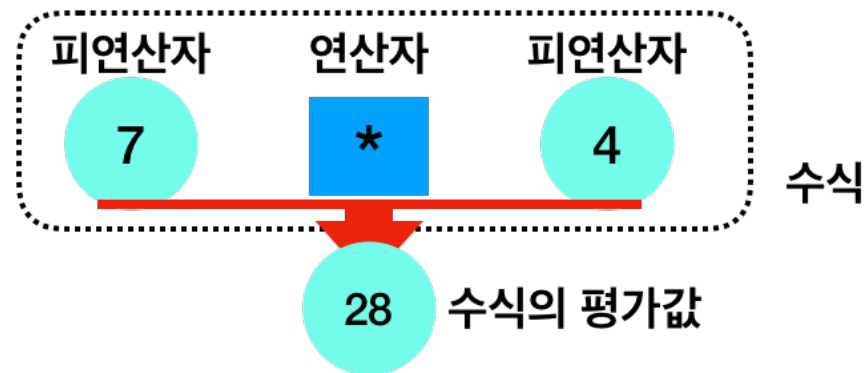


잠깐 - 할당 연산자와 변수의 선언

전통적으로 프로그래밍에서 어떤 변수를 사용하려고 하면, 변수의 이름과 자료형을 밝히고 앞으로 이 변수를 사용하겠다는 **선언** `declaration`을 해야 한다. 그런데 파이썬은 변수를 선언하지 않고 바로 사용할 수 있다. 우리도 지금까지 변수를 선언한 적이 없다. 하지만 내부적으로는 변수를 선언하는 일이 이루어지는데, 어떤 변수에 할당 연산자가 처음으로 적용되는 순간 우리에게 보이지 않는 방식으로 선언이 이루어진다. 이때 파이썬은 할당 연산자의 오른쪽 수식이 나타내는 값에 따라 변수의 자료형을 결정하게 된다.

3.3 수식과 연산자는 어떻게 쓰는 걸까

- 수식 `expression`이란 피연산자들과 연산자의 조합이라고 할 수 있다. 이때 연산자 `operator`는 어떤 연산을 나타내는 기호를 의미하며, 피연산자 `operand`는 연산의 대상이 되는 숫자나 변수를 의미한다.
- 아래의 수식 $7 + 4$ 에서 7과 4는 피연산자이고 덧셈을 의미하는 $+$ 는 연산자이다.
- 이때 연산자와 피연산자 사이에 공백을 삽입하는 것이 파이썬의 코딩 관습에 더 적합한 방식이다. 즉 $a*b$ 보다 $a * b$ 로 코딩하는 것이 좋다.



3.3 수식과 연산자는 어떻게 쓰는 걸까

- 산술 연산자는 덧셈, 뺄셈, 곱셈, 나눗셈과 나머지 연산을 실행하는 여러가지 연산자이다. 아래 표에 산술 연산자들과 그 사용예, 그리고 결과값을 정리하였다.

연산자	기호	사용예	결과값
덧셈	+	$7 + 4$	11
뺄셈	-	$7 - 4$	3
곱셈	*	$7 * 4$	28
지수	**	$7 ** 2$	$7^2 = 49$
나눗셈(정수 나눗셈의 몫)	//	$7 // 4$	1
나눗셈(실수 나눗셈)	/	$7 / 4$	1.75
나머지	%	$7 \% 4$	3

3.3 수식과 연산자는 어떻게 쓰는 걸까

- 덧셈, 뺄셈, 곱셈은 결과를 이해하는 데에 큰 어려움이 없을 것이다. 나눗셈 연산시 주의할 점이 있는데, 파이썬에서 나눗셈 연산 / 은 항상 실수로 계산된다는 점이다.

```
>>> 7 / 4          # 나눗셈은 항상 실수로 계산된다
1.75
>>> 8 / 4          # 나눗셈이기 때문에 정수 2가 아닌 실수 2.0이 출력
2.0
```

- 하지만 경우에 따라서는 나눗셈의 몫을 정수로 계산하고 싶은 경우가 있을 것이다. 이때는 //을 사용한다. //을 사용하여 나눗셈을 하면 소수점 이하는 없어지고 정수 부분만 남는다.

```
>>> 7 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
1
>>> 8 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
2
>>> 9 // 4         # //을 사용해서 나누면 정수 몫만 나온다.
2
```

3.4 컴퓨터과학에서 아주 중요한 나머지 연산자 : %

- 프로그래밍에서 놀라울 정도로 많이 사용되는 연산자가 나머지 연산자(%)이다. $x \% y$ 는 x 를 y 로 나누어서 남은 나머지를 반환한다.
- 예를 들어 $11 \% 4$ 은 3이다. 아래 그림을 보면 11 송이의 꽃이 있는데, 이것을 네 명에게 똑같은 갯수로 나누어 주면 2 송이씩 줄 수 있다. 그러면 나누어 주고 남은 꽃이 3 송이가 된다. $11 \% 4$ 의 값은 이 남은 꽃의 수 3이 되는 것이다.

11 % 4



3.4 컴퓨터과학에서 아주 중요한 나머지 연산자 : %

- 몫은 // 연산자가 계산할 수 있고 나머지는 % 연산자로 계산할 수 있다. 예를 들어서 7 / 4 연산에서 몫과 나머지를 계산하는 코드는 다음과 같다. 이 때 입력값으로 실수값도 가능하다.

```
p = int(input("분자를 입력하시오: "))  
q = int(input("분모를 입력하시오: "))  
print("나눗셈의 몫 =", p // q)  
print("나눗셈의 나머지 =", p % q)
```

```
분자를 입력하시오: 7  
분모를 입력하시오: 4  
나눗셈의 몫 = 1  
나눗셈의 나머지 = 3
```

3.4 컴퓨터과학에서 아주 중요한 나머지 연산자 : %

- 그런데 나머지 연산자를 어디에 이용하면 좋을까?
- 나머지 연산자를 이용하면 짝수와 홀수를 쉽게 구분할 수 있다. 즉 어떤 수 x 를 2로 나누어서 나머지가 0이면 짝수이다.
- 아래와 같은 코드를 작성하여 실행해 보자. 이 코드는 정수를 하나 입력하도록 요구하고, 입력된 정수가 짝수이면 0, 홀수이면 1을 출력할 것이다.

```
number = int(input("정수를 입력하시오: "))  
print(number % 2)    # 나머지로 연산의 결과가 0이면 짝수, 1이면 홀수이다
```

```
정수를 입력하시오: 28  
0
```




도전문제 3.1

사용자가 시간을 초를 입력하면 입력된 초가 몇 시간 몇 분 몇 초에 해당하는지 다음과 같이 출력하는 코드를 뭉과 나머지 연산을 이용해 구현해 보아라.

초를 입력하세요: 2323423

입력한 시간은 645 시간 23 분 43 초입니다.

3.5 거듭제곱 연산자 : **

- 파이썬은 대단히 친절한 프로그래밍 언어이다. 우리에게 익숙한 **거듭제곱**^{power} 계산도 연산자로 제공한다.
- 거듭제곱을 계산하려면 ** 연산자를 사용한다. 예를 들어서 2^7 을 계산하는 파이썬 수식은 `2 ** 7`이다.
- 이때 2는 밑(base)이라고 하고 7을 지수(exponent)라고 한다.

```
>>> 2 ** 7  
128
```

```
# 2의 7제곱이 계산되며 2*2*2*2*2*2*2와 동일함
```

3.5 거듭제곱 연산자 : **

- 수학에서처럼 거듭제곱 연산자는 다른 연산자들보다 높은 우선순위를 가진다.
- 예를 들어서 $10*2**7$ 은 20^7 이 아니라 10×2^7 으로 계산되어 1280이다. 다른 연산자와 달리 거듭제곱 연산자들은 오른쪽에서 왼쪽으로 계산된다.
- 예를 들어서 $2**2**3$ 은 $(2**2)**3$ 가 아니라 $2**(2**3)$ 로 계산되어 $2**8$ 과 같은 256이다.

3.5 거듭제곱 연산자 : **

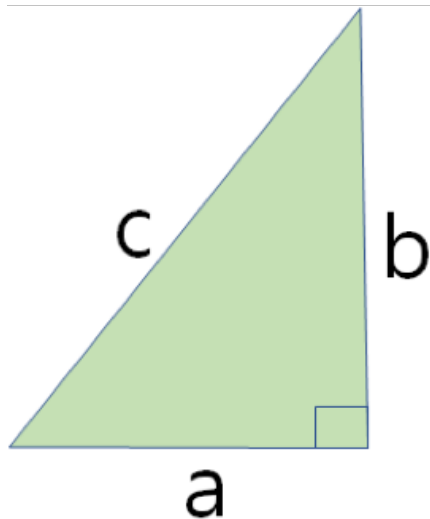
- 원리금 합계를 복리로 계산하는 식을 파이썬으로 만들어 보자.
- 원금이 a , 이자율이 r 일 경우, n 년 후의 원리금 합계는 $a(1 + r)^n$ 이 된다.
- 이것을 파이썬 코드를 작성하여 계산하면 다음과 같다.

```
a = 1000
r = 0.05
n = 10
print(a * (1 + r) ** n)
```

```
1628.894626777442
```

3.5 거듭제곱 연산자 : **

- 지수 연산자를 이용하면 제곱근이나 세제곱근도 쉽게 계산할 수 있다. 제곱근은 지수가 1/2인 값이기 때문이다. 예를 들어 다음과 같은 직각삼각형이 있다고 생각해 보자.



Pythagoras (Πυθαγόρας)

기원전 570년–기원전 495년

$$c^2 = a^2 + b^2$$

제가 발견한 정리라고 합니다.

정말일까요?

저의 제자가 발견했다고도 하죠.

3.5 거듭제곱 연산자 : **

- 피타고라스 정리에 의해 직각삼각형의 빗변의 길이 c 의 제곱은 밑변 길이 a 의 제곱과 높이 b 의 제곱을 더한 것과 같다는 것을 잘 알고 있을 것이다.
- 그러면 밑변과 높이를 입력하여 빗변을 계산하는 프로그램을 작성하려면 어떻게 해야 할까? 식을 조금만 변경하면 다음과 같다는 것을 알 수 있다.

$$c = \sqrt{a^2 + b^2} = (a^2 + b^2)^{\frac{1}{2}}$$

3.5 거듭제곱 연산자 : **

- 따라서 다음과 같이 프로그램을 작성하면 된다.

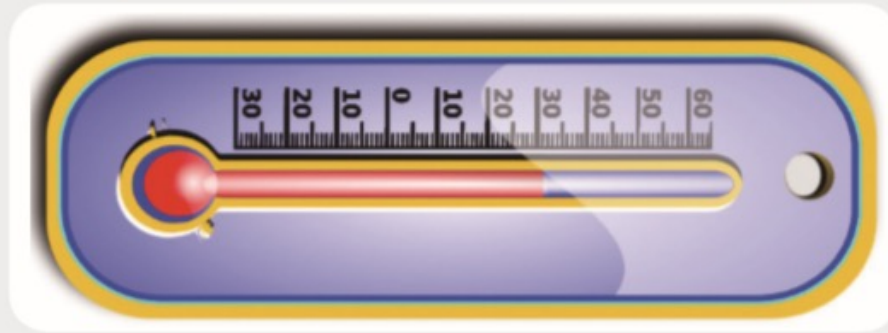
```
bottom = float(input('직각삼각형의 밑변의 길이를 입력하시오: '))  
height = float(input('직각삼각형의 높이를 입력하시오: '))  
hypotenuse = (bottom ** 2 + height ** 2) ** 0.5  
print('빗변은', hypotenuse, '입니다')
```

```
직각삼각형의 밑변의 길이를 입력하시오: 3  
직각삼각형의 높이를 입력하시오: 4  
빗변은 5.0 입니다
```

LAB³⁻² 화씨온도를 섭씨온도로 변환하기

우리나라에서는 온도를 나타낼 때 섭씨온도를 사용하지만 미국에서는 **화씨온도**fahrenheit를 사용한다. 화씨온도를 받아서 섭씨온도로 바꾸는 프로그램을 작성해보자. 화씨온도를 F 라고 하면 섭씨온도 C 로 바꾸는 식은 다음과 같다.

$$C = (F - 32) \times \frac{5}{9}$$



원하는 결과

화씨온도: 100

섭씨온도: 37.7777777777778

LAB³⁻² 화씨온도를 섭씨온도로 변환하기

```
fahrenheit = int(input("화씨온도: "))  
celsius = (fahrenheit - 32) * 5 / 9  
print("섭씨온도:", celsius)
```

혹은 아래와 같이 실수로 처리해도 된다. 위의 코드가 제대로 동작한 이유는 나눗셈이 사용되면 무조건 실수로 바뀌게 되고, 정수와 실수의 곱도 실수로 간주되게 때문에 최종적으로 실수 값이 나오게 된다.

```
fahrenheit = float(input("화씨온도: "))  
celsius = (fahrenheit - 32.0) * 5.0 / 9.0  
print("섭씨온도:", celsius)
```

LAB³⁻³ 몸무게와 키를 입력받아 BMI 계산하기

BMIBody Mass Index는 체중(kg)을 신장(m)의 제곱으로 나눈 값으로 체지방 축적을 잘 반영하기 때문에 비만도 판정에 많이 사용한다. 앞 장에서도 BMI 계산을 해보았지만, 변수에 값을 바로 지정해서 결과를 얻었다. 이번에는 사용자의 입력을 받아 처리하도록 개선해 보자. BMI 지수는 킬로그램 단위로 측정한 체중을 w , 미터 단위로 측정한 키가 h 라고 할 때 다음과 같이 구할 수 있다.

$$BMI = \frac{w}{h^2}$$

원하는 결과

몸무게를 kg 단위로 입력하시오: 95

키를 미터 단위로 입력하시오: 1.82

당신의 BMI= 28.680111097693512

LAB³⁻³ 몸무게와 키를 입력받아 BMI 계산하기

```
weight = float(input("몸무게를 kg 단위로 입력하시오: "))  
height = float(input("키를 미터 단위로 입력하시오: "))  
bmi = (weight / (height ** 2))  
print("당신의 BMI=", bmi)
```



도전문제 3.2

BMI는 비만도를 추정하는 데에 가장 흔히 사용되는 지수이지만, 정확도에 한계가 있다는 비판이 많다. BMI 처럼 간단하면서 비만도 계산의 정확도가 높은 것으로 알려진 새로운 지수가 있다. **상대적 지방 질량** **relative fat mass** 지수로 간단히 줄여서 **RFM**이라고 한다. 남자와 여자에 적용되는 수식이 다르다.

여성을 위한 공식: $76 - (20 \times (\text{신장}/\text{허리둘레}))$

남성을 위한 공식: $64 - (20 \times (\text{신장}/\text{허리둘레}))$

여러 논문을 통해 오른쪽 그림처럼 RFM 지수가 BMI 지수보다 실제 체지방 비율과 더 잘 일치한다고 보고되었다.

남녀 모두를 위한 공식은 성별에 여성은 1, 남성은 0을 적용하여 다음과 같이 구할 수 있다.

$$\text{RFM} = 64 - (20 \times (\text{신장} / \text{허리둘레})) + 12 \times \text{성별}$$

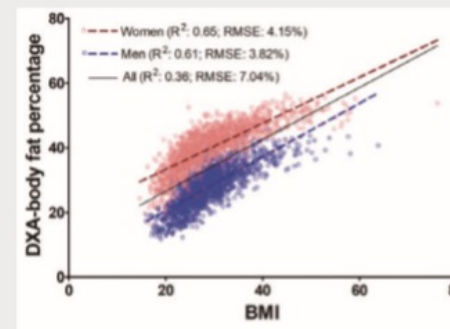
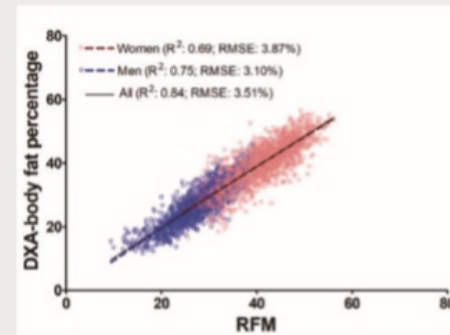
RFM을 계산하는 프로그램을 다음과 같은 방식으로 동작하게 만들어 보라.

여성이면 1, 남성이면 0을 입력하세요: 0

당신의 키는 얼마입니까? 181

당신의 허리 둘레는 얼마입니까? 94

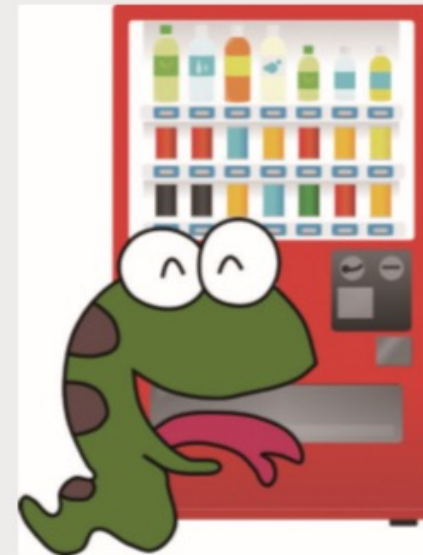
당신의 RFM은 25.48936170212766



출처: Woolcott OO, Bergman RN (2018) Relative fat mass (RFM) as a new estimator of whole-body fat percentage horizontal line – A cross-sectional study in American adult individuals. Scientific Reports 8(1): 1–11, 2018.

LAB³⁻⁴ 자동 판매기 프로그램을 만들어 보자

자동 판매기를 시뮬레이션하는 프로그램을 작성해보자. 자동 판매기는 사용자로부터 투입한 돈과 물건값을 입력받는다. 물건값은 100원 단위라고 가정한다. 프로그램은 잔돈을 계산하여 출력한다. 자판기는 동전 500원, 100원짜리만 가지고 있다고 가정하자. 투입한 금액과 물건값을 입력으로 받아 거스름돈으로 500원 동전 몇 개와 100원 동전 몇 개를 내어 보내면 되는지 계산하는 프로그램을 작성하라.



원하는 결과

투입한 돈: 5000

물건값: 2600

거스름돈: 2400

500원 동전의 개수: 4

100원 동전의 개수: 4

LAB³⁻⁴ 자동 판매기 프로그램을 만들어 보자

```
money = int(input("투입한 돈: "))
price = int(input("물건값: "))

change = money - price
print("거스름돈: ", change)
coin500s = change // 500      # 500으로 나누어서 몫이 500원 동전의 개수
change = change % 500        # 500으로 나눈 나머지를 계산한다.
coin100s = change // 100     # 100으로 나누어서 몫이 100원짜리의 개수

print("500원 동전의 개수:", coin500s)
print("100원 동전의 개수:", coin100s)
```

3.6 복합 할당 연산자라는 편리한 연산자

- 산술 연산자와 할당 연산자를 결합하여 조금 간략하게 표현하는 방법이 있다.
- 만일 특정한 산술 연산자를 @라고 하면 복합 할당 연산자는 @= 모양이 되면서 다음과 같은 동작을 한다.

$$a \ @ = \ b \quad \Longleftrightarrow \quad a \ = \ a \ @ \ b$$

3.6 복합 할당 연산자라는 편리한 연산자

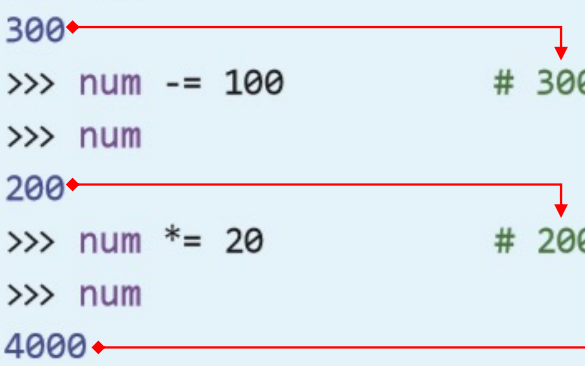
- 파이썬은 다음과 같은 다양한 복합 할당 연산자를 지원한다. 복합 할당 연산자를 모른다고 하여도 원래의 산술 연산자와 할당 연산자를 이용하여 아무런 문제 없이 코딩을 할 수 있다.
- 하지만, 프로그래머들이 흔히 쓰는 이러한 복합 할당 연산자를 전혀 모른다면 다른 사람의 코드를 읽는 데에 문제가 있을 수 있으므로 잘 이해해 놓도록 하자.

연산자	사용 방법	의미
<code>+=</code>	<code>i += 10</code>	<code>i = i + 10</code>
<code>-=</code>	<code>i -= 10</code>	<code>i = i - 10</code>
<code>*=</code>	<code>i *= 10</code>	<code>i = i * 10</code>
<code>/=</code>	<code>i /= 10</code>	<code>i = i / 10</code>
<code>//=</code>	<code>i //= 10</code>	<code>i = i // 10</code>
<code>%=</code>	<code>i %= 10</code>	<code>i = i % 10</code>
<code>**=</code>	<code>i **= 10</code>	<code>i = i ** 10</code>

3.6 복합 할당 연산자라는 편리한 연산자

- 복합 할당 연산자의 동작을 이해하기 위해 아래와 같이 인터프리터에서 다양한 복합 할당 연산자를 연습해 보면 어렵지 않게 이해할 수 있을 것이다.

```
>>> num = 200
>>> num += 100          # 200 + 100 연산을 수행하여 그 결과를 num에 할당
>>> num
300
>>> num -= 100          # 300 - 100 연산을 수행하여 그 결과를 num에 할당
>>> num
200
>>> num *= 20            # 200 * 20 연산을 수행하여 그 결과를 num에 할당
>>> num
4000
>>> num /= 2             # 4000 / 2 연산을 수행하여 그 결과를 num에 할당
>>> num
2000.0
```



3.7 두 값의 크기를 비교하는 비교 연산자

- 다음은 **비교 연산자** `comparison operator`에 대해 알아보자. '크다' 혹은 '작다'와 같은 비교 연산은 수치 데이터를 담고 있는 두 개의 피연산자를 대상으로 크기 관계를 살펴본다.
- 그리고 그 결과인 True 혹은 False를 반환한다. 이렇게 True나 False의 값을 갖는 자료형을 **부울** `bool`형이라고 한다.

3.7 두 값의 크기를 비교하는 비교 연산자

- 파이썬은 아래와 같은 비교 연산자를 제공하고 있다. 가장 오른쪽 열에는 a와 b의 값이 각각 100과 200이라고 가정하고, 해당 비교 연산자가 적용되었을 때 얻게 되는 부울형 값을 보이고 있다.

연산자	설명	a = 100 b = 200
==	두 피연산자의 값이 같으면 True를 반환	a == b는 False
!=	두 피연산자의 값이 다르면 True를 반환	a != b는 True
>	왼쪽 피연산자가 오른쪽 피연산자보다 클 때 True를 반환	a > b는 False
<	왼쪽 피연산자가 오른쪽 피연산자보다 작을 때 True를 반환	a < b는 True
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같을 때 True 반환	a >= b는 False
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같을 때 True 반환	a <= b는 True

3.7 두 값의 크기를 비교하는 비교 연산자

- == 연산자는 값이 동일할 때만 True를 반환한다. 예를 보면 a == b문에서 변수 a는 100, b는 200으로 값이 같지 않다. 따라서 False를 반환한다.
- 두 번째 비교 연산자(!=)는 값이 동일하지 않을 때만 True를 반환한다. 따라서 예를 보면 변수 a와는 b는 100과 200으로 같지 않다. 따라서 a != b는 결과로 True를 반환한다.
- 이처럼 비교 연산자는 크다, 작다, 크거나 같다, 작거나 같다 등 값의 비교를 할 수 있도록 해준다.

3.7 두 값의 크기를 비교하는 비교 연산자

- 실습으로 확인해 보자.

```
>>> a, b = 100, 200      # a에는 100, b에는 200을 할당하는 동시할당문
>>> a == b
False
>>> a != b
True
>>> a > b
False
>>> a < b
True
>>> a >= b
False
>>> a >= b              # 두 개의 기호로 표현된 비교 연산자는 띄어쓰면 안 된다 : ERROR
File "<stdin>", line 1
a > = b
^
SyntaxError: invalid syntax
>>> a => b              # 두 개 기호로 표현된 비교 연산자의 순서를 뒤집어도 안 된다 : ERROR
File "<stdin>", line 1
a => b
^
SyntaxError: invalid syntax
```



잠깐 - 두 개의 문자로 된 비교 연산자는 그 묶음 그대로가 하나의 연산자이다

비교 연산자 중에는 != 혹은 >= 와 같이 두 개의 문자가 사용되는 경우가 있다. 이 연산자들은 띄어쓰거나 순서를 바꾸면 안 된다. != 에서 !와 = 사이에 공백을 넣으면 안되며, >= 연산에서도 >와 = 사이에 공백을 넣으면 안된다. 그리고 => 과 같이 등호와 > 연산자의 순서가 바뀌어도 에러가 뜬다.

3.8 AND, OR, NOT도 연산자로 사용가능하다 : 논리 연산자

- 파이썬은 and, or, not 연산자를 지원하는데 이러한 연산자를 **논리 연산자**logical operator라고 한다. 이 연산은 논리 and, or, not 연산을 통해 True나 False 중 하나의 값을 가지는 **부울**bool 값을 반환한다.
- 부울 자료형은 True와 False 값을 가지는 자료형인데, 원래 부울 값이 아닌 자료형의 데이터도 부울형으로 변환될 수 있다.
- 먼저 숫자 0을 bool 형으로 변환하면 False가 되고, 0을 제외한 모든 수를 bool 형으로 변환하면 True가 된다.

3.8 AND, OR, NOT도 연산자로 사용가능하다 : 논리 연산자

- 그렇다면 문자열은 어떨까? 빈 문자열은 False, 빈 문자열을 제외한 모든 문자열은 True로 간주되어 논리 연산에 활용될 수 있다.

```
>>> 10 > 20      # 10은 20보다 작으므로 10 > 20은 False가 됨
False
>>> 10 < 20      # 10은 20보다 작으므로 10 < 20은 True가 됨
True
>>> bool(9)       # 9는 0이 아니므로 True가 됨
True
>>> bool(-1)      # -1 역시 0이 아니므로 True가 됨
True
>>> bool(0)       # 숫자 값중에서는 유일하게 0의 값만 False가 됨
False
>>> bool(None)    # None은 값이 없음을 표현함, 따라서 False가 됨
False
>>> bool('')      # 빈 문자열이므로 False가 됨
False
>>> bool('hello') # 문자열 값이 있으므로 True가 됨
True
```


3.8 AND, OR, NOT도 연산자로 사용가능하다 : 논리 연산자

- 앞의 실습 결과를 보면 9와 -1은 True, 0은 False로 간주되는 것을 볼 수 있다. **None**이나 아무 것도 들어있지 않은 문자열 (") 또한 False로 간주되는 것을 살펴볼 수 있다.
- 여기서 **None**이란 값이 없는 것을 표현하는 파이썬의 키워드이다. 0이나 비어 있는 것은 False, 0이 아닌 값이나 무엇인가 들어있는 것은 True로 간주된다는 것을 잘 확인해 두자.

3.8 AND, OR, NOT도 연산자로 사용가능하다 : 논리 연산자

- 이러한 부울값을 가진 데이터에 대해서 적용할 수 있는 연산이 논리 연산이다. 논리 연산은 부울형 자료의 값을 조합하여 새로운 부울값을 만들어 내는 것이다.
- 파이썬의 논리 연산자는 다음과 같은 것들이 있다.

연산자	의미
x and y	x와 y중 거짓(False)이 하나라도 있으면 거짓이 되며 모두 참(True)인 경우에만 참이다.
x or y	x나 y중에서 하나라도 참이면 참이 되며, 모두 거짓일 때만 거짓이 된다.
not x	x가 참이면 거짓, x가 거짓이면 참이 된다.

3.8 AND, OR, NOT도 연산자로 사용가능하다 : 논리 연산자

- x 와 y 가 가진 값에 따라 논리 연산자가 반환하는 값을 정리한 결과는 아래 그림과 같다.

x and y

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

x or y

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

not x

x	not x
False	True
True	False

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 사칙연산과 같은 산술 연산자와 비교 연산과 같은 논리 연산자 처럼 연산의 대상이 되는 데이터의 값을 가지고 처리하는 연산자가 있는가 하면, 저장된 정보의 각 **비트**bit 단위로 처리가 이루어지는 연산도 있다.
- 컴퓨터는 정보를 0과 1로 구성된 2진수를 사용하며, 한 자리의 이진수를 **비트**bit 라고 한다. 정수 데이터형에 대하여 비트 단위의 조작이 가능한데 이 조작을 위한 연산자가 **비트 연산자**bit operator 이다.
- 비트 연산자는 다른 말로 **비트 단위 연산자**bitwise operator라고도 한다. 비트 단위 연산자는 정수형의 피연산자에만 적용할 수 있다.

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 사용 가능한 비트 연산자는 다음과 같은 것들이 있다.

연산자	의미	설명
&	비트 단위 AND	두 개의 피연산자의 해당 비트가 모두 1이면 1, 아니면 0
	비트 단위 OR	두 피연산자의 해당 비트 중 하나라도 1이면 1, 아니면 0
^	비트 단위 XOR	두 개의 피연산자의 해당 비트의 값이 같으면 0, 아니면 1
~	비트 단위 NOT	0은 1로 만들고, 1은 0으로 만든다.
<<	비트 단위 왼쪽으로 이동	지정된 개수만큼 모든 비트를 왼쪽으로 이동시킨다.
>>	비트 단위 오른쪽으로 이동	지정된 개수만큼 모든 비트를 오른쪽으로 이동시킨다.

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 연산자 &는 두 개의 피연산자가 가진 비트들을 하나씩 짝을 지어 각각에 대해 연산을 적용한다. 짝이 되는 비트가 모두 1이면 1, 아니면 0을 반환한다. 1을 True, 0을 False로 생각할 때 이 연산자는 각 비트에 대해 and 연산자와 비슷한 일을 수행한다.
- 마찬가지로 연산자 |는 비트 단위로 OR 연산을 수행하고, ~ 연산자는 비트 단위로 NOT 연산을 수행한다.

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 어떤 정수의 이진수 값을 확인하고 싶으면 `bin()` 함수를 사용하면 된다. 이를 이용해서 비트 연산자의 동작을 확인해 보자.

```
>>> bin(9)          # 2진수 형식 출력 (00001001)
'0b1001'
>>> bin(10)         # 2진수 형식 출력 (00001010)
'0b1010'
>>> bin(9 & 10)     # 9와 10을 2진수로 표현했을 때 모두 1인 위치만 1 나머지는 0이 된다
'0b1000'
>>> bin(9 | 10)     # 9와 10을 2진수로 표현했을 때 하나로도 1이 나타나는 자리는 1이 된다
'0b1011'
```

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- \wedge 연산자는 해당 비트의 값이 같으면 0, 아니면 1을 반환하는 XOR 연산을 비트 단위로 수행한다.
- 지금 당장은 XOR는 데이터를 활용하는 방법에 익숙하지 않겠지만, XOR 연산은 데이터를 교환하거나, 특정 부분을 제거하는 일들을 할 때 유용하게 사용된다.

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 숫자 9와 10에 대해 비트 단위 XOR 연산을 수행하면 비교하는 두 개의 비트 중에 값이 서로 다른 경우에만 1이 된다. 따라서 $9 \wedge 10$ 의 결과는 2진수 11이 되고, 10진수로는 3이 된다.

```
>>> bin(9)          # 이진수 00001001
'0b1001'
>>> bin(10)         # 이진수 00001010
'0b1010'
>>> 9 ^ 10          # 결과는 00000011 = 십진수 3
3
>>> bin(9 ^ 10)     # 앞 부분의 0은 모두 사라지고 1이 나타나는 곳부터 출력된다.
'0b11'
```

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- << 연산자는 지정된 수만큼 모든 비트를 왼쪽을 이동시키는 연산자이고, >> 연산자는 반대로 지정된 수만큼 모든 비트를 오른쪽으로 이동시키는 연산자이다.
- 비트가 한칸 왼쪽으로 이동하면 정수는 2배의 값을 가지게 되고, 한 칸 오른쪽으로 이동하면 //2 연산을 한 결과가 될 것이다. 이런 연산은 **쉬프트**shift 연산이라고 한다.

```
>>> 4 << 1      # 00100을 한 비트 왼쪽으로 이동하여 01000을 만든다.  
8  
>>> 4 << 2      # 00100을 두 비트 왼쪽으로 이동하여 10000을 만든다.  
16
```

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 앞서 다룬 복합 할당 연산자는 비트 단위 연산자에도 적용할 수 있다. 따라서 다음과 같은 비트 단위 복합 할당 연산자도 사용할 수 있다.

연산자	사용 예시	의미
<code>&=</code>	<code>i &= 10</code>	<code>i = i & 10</code>
<code> =</code>	<code>i = 10</code>	<code>i = i 10</code>
<code>^=</code>	<code>i ^= 10</code>	<code>i = i ^ 10</code>
<code><<=</code>	<code>i <<= 10</code>	<code>i = i << 10</code>
<code>>>=</code>	<code>i >>= 10</code>	<code>i = i >> 10</code>

3.9 이진수를 잘 다루는 컴퓨터에 최적화된 연산자 : 비트 연산자

- 2의 거듭제곱수를 계속해서 만들어 보는 일은 다음과 같은 코드로 가능할 것이다. 마지막으로 정수 8에 $\gg= 1$ 연산을 적용하면 그 결과는 4가 되는 것도 확인 할 수 있다.

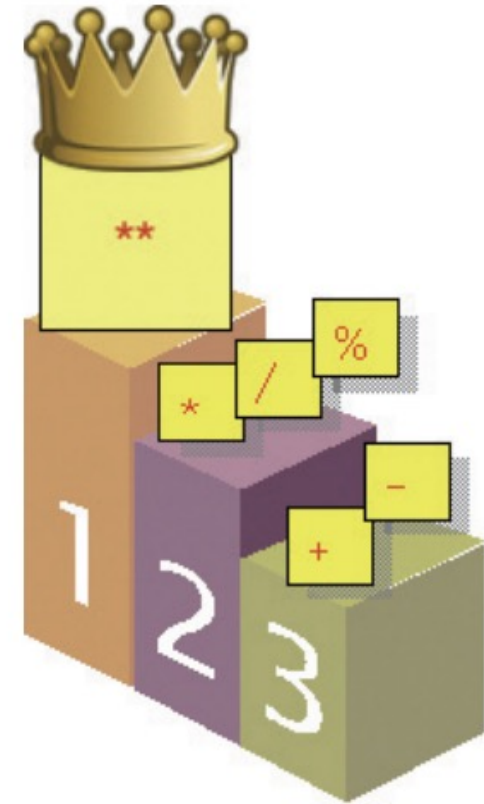
```
>>> num = 2      # 정수 2는 이진수 10(2)으로 표현됨
>>> num
2
>>> num <<= 1    # 정수 2를 1비트 왼쪽 이동시키면 100(4)가 됨
>>> num
4
>>> num <<= 1    # 정수 4를 1비트 더 왼쪽 이동시키면 1000(8)이 됨
>>> num
8
>>> num >>= 1    # 정수 8를 1비트 오른쪽으로 이동시키면 100(4)가 됨
>>> num
4
```

3.10 연산자 사이에도 먼저 처리하는 것이 있다

- 만약 아래와 같이 하나의 수식이 2개 이상의 연산자를 가지고 있는 경우에는 어떤 연산자가 먼저 수행될 것인가? 예를 들면 다음과 같은 문장에서 가장 먼저 수행되는 연산은 무엇인가?

$$x + y * z$$

Diagram illustrating operator precedence in the expression $x + y * z$. A bracket labeled ① groups the multiplication $y * z$, and a larger bracket labeled ② groups the entire expression $x + y * z$, indicating that multiplication is performed first.



3.10 연산자 사이에도 먼저 처리하는 것이 있다

- 우리가 수학 시간에 배웠듯이 곱셈과 나눗셈이 덧셈과 뺄셈보다 먼저 수행되어야 한다. 이를 결정하는 것이 우선순위이다.
- 이것은 많은 연산들 중에서 어떤 연산을 먼저 수행할지를 결정하는 규칙이다. 각 연산자들은 서열이 매겨져 있다. 즉 곱셈과 나눗셈은 덧셈이나 뺄셈보다 우선순위가 높다.
- 만약 사용자가 이러한 우선순위대로 연산을 하지 않고 다른 순서로 하고 싶은 경우는 어떻게 하면 되는가? 수학 시간에 사용한 수식처럼 이 경우에는 **괄호를 사용**하면 된다.

$$\underbrace{x + \underbrace{y * z}_1}_{2}$$

$$\underbrace{\underbrace{(x + y)}_1 * z}_2$$

3.10 연산자 사이에도 먼저 처리하는 것이 있다

연산자	설명
**	지수 연산자
~ , + , -	단항 연산자
* , / , % , //	곱셈, 나눗셈, 나머지 연산자
+ , -	덧셈, 뺄셈
>> , <<	비트 이동 연산자
&	비트 AND 연산자
^ ,	비트 XOR 연산자, 비트 OR 연산자
<= , < , > , >=	비교 연산자
== , !=	동등 연산자
= , %= , /= , //= , -= , += , *= , **=	할당 연산자, 복합 할당 연산자
is , is not	아이덴티티 연산자
in , not in	소속 연산자
not , or , and	논리 연산자

높은
우선순위



LAB³⁻⁵ 평균 구하기 - 연산자 우선순위

수식을 프로그램으로 구현하는 경우에는 연산자의 우선순위에 주의할 필요가 있다. 사용자로부터 3개의 수를 입력받아서 평균값을 계산하여 출력하는 프로그램을 살펴보자. 이것을 다음과 같이 코딩하면 안 된다.

```
x = int(input("첫 번째 수를 입력하시오: "))
y = int(input("두 번째 수를 입력하시오: "))
z = int(input("세 번째 수를 입력하시오: "))
```

```
avg = x + y + z / 3    # 잘못된 코딩
print("평균 =", avg)
```

이렇게 코딩을 하면 10, 20, 30을 입력했을 때에 평균이 옳은 값인 20이 나오지 않고 40으로 잘못 계산되어 나온다. 우리가 원하는 결과는 아래와 같다. 어디를 고쳐야 할지 생각해 보고 수정해서 결과를 확인하라.

원하는 결과

```
첫 번째 수를 입력하시오: 10
두 번째 수를 입력하시오: 20
세 번째 수를 입력하시오: 30
평균 = 20.0
```


LAB³⁻⁵ 평균 구하기 - 연산자 우선순위

```
x = int(input("첫 번째 수를 입력하시오: "))  
y = int(input("두 번째 수를 입력하시오: "))  
z = int(input("세 번째 수를 입력하시오: "))  
  
avg = (x + y + z) / 3    # 올바르게 고쳐진 계산  
print("평균 =", avg)
```



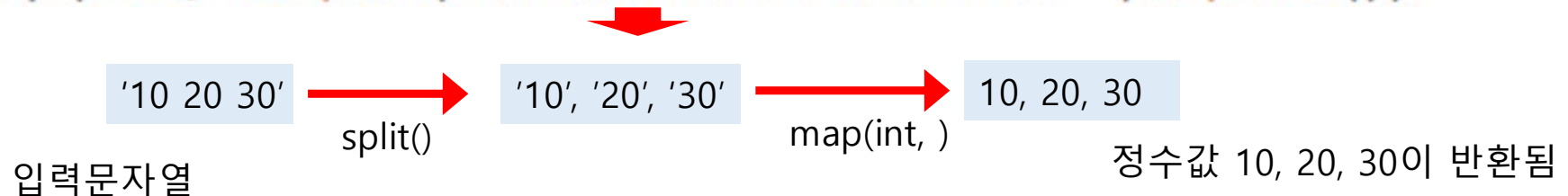
잠깐 - 사용자로부터 여러 개의 수를 입력받자.

사용자로 부터 개별적인 수를 여러줄에 걸쳐 입력받을 수도 있으나 다음과 같은 코드를 사용하면 한 줄에 3개의 정수를 한꺼번에 입력받을 수 있다.

```
>>> x, y, z = map(int, input('세 정수를 입력하시오: ').split())
세 정수를 입력하시오: 10 20 30
>>> x, y, z
(10, 20, 30)
```

split() 메소드는 입력된 문자열을 공백단위로 나누어서 '10' 과 '20', '30'의 세 문자열로 나누어 주며 이를 map() 함수가 int형으로 바꾸어 주는 일을 한다.

`map(int, input('세 정수를 입력하시오: ').split())`



3.11 랜덤 모듈과 math 모듈로 다양한 기능을 사용해보자

- 파이썬은 개발자들이 손쉽게 프로그램을 작성할 수 있도록 다양한 내장함수를 제공하는데 `print()`, `input()`, `len()`, `sum()`, `min()`, `max()`, `int()`, `str()` 등 70여 가지 이상을 제공하고 있다.
- 이뿐 아니라 표준 라이브러리라는 이름으로 난수 생성기와 다양한 수학 함수 등의 기능을 제공하고 있다.
- 이 표준 라이브러리 역시 워낙 방대하기 때문에 여기에서는 난수와 관련한 함수를 제공하는 `random` 모듈과 수학 함수를 제공하는 `math` 모듈에 대하여 간단하게 설명한다.

3.11 랜덤 모듈과 math 모듈로 다양한 기능을 사용해보자

- random 모듈은 임의의 수를 생성하거나, 리스트나 튜플내의 요소를 무작위로 섞거나, 선택하는 함수를 포함하고 있다. random 모듈에 포함되어 있는 대표적인 함수 몇 가지를 함께 알아보자.

```
>>> import random
>>> random.random()          # 0 이상 1 미만의 임의의 실수를 반환
0.19452357419514088
>>> random.random()          # 이 함수는 매번 다른 실수를 반환
0.6947454047320903
>>> random.randint(1, 7)      # 1 이상 7 이하(7을 포함)의 임의의 정수를 반환
3
>>> random.randrange(7)       # 0 이상 7 미만(7을 포함하지 않음)의 임의의 정수를 반환
3
>>> random.randrange(1, 7)    # 1 이상 7 미만(7을 포함안함)의 임의의 정수를 반환
6
>>> random.randrange(0, 10, 2) # 0, 2, 4, 8 중(10은 포함 안함) 하나를 반환함
2
>>> lst = [10, 20, 30, 40, 50] # 여러개의 값을 가지는 리스트를 생성함
>>> random.shuffle(lst)        # lst 리스트의 순서를 무작위로 섞음
>>> lst
[40, 50, 10, 20, 30]
>>> random.choice(lst)        # lst 리스트의 원소 중 무작위로 하나를 고름
30
```

3.11 랜덤 모듈과 math 모듈로 다양한 기능을 사용해보자

- math 모듈은 원주율 π , 자연상수 e 등의 상수 값과 실수의 절대값 `fabs()`, `ceil()`, `floor()`, `log()`, `sin()`, `cos()` 등의 다양한 수학 함수를 포함하고 있으며 다음과 같이 사용할 수 있다.

```
>>> import math
>>> math.pow(3, 3)      # 3의 3 제곱
27.0
>>> math.fabs(-99)      # -99의 실수 절대값
99.0
>>> math.log(2.71828)
0.9999999327347282
>>> math.log(100, 10)   # 로그 10을 밑으로 하는 100값
2
>>> math.pi            # 원주율
3.141592653589793
>>> math.sin(math.pi / 2.0)  # sin() 함수의 인자로 PI/2.0를 넣어보자
1.0
```



summary

핵심 정리



- 수식은 피연산자와 연산자로 이루어진다.
- 덧셈, 뺄셈, 곱셈, 나눗셈을 위하여 $+$, $-$, $*$, $/$ 기호를 사용한다.
- 지수 연산자는 $**$ 이다.
- 나눗셈에서 몫을 계산하려면 $//$ 연산자를 사용한다.
- 나눗셈에서 나머지를 계산하려면 $%$ 연산자를 사용한다.
- 복합 연산자는 할당 연산자와 각종 산술 연산자를 합쳐놓은 것이다.
- 우선순위가 높은 연산자가 먼저 계산되며, 연산자의 우선 순서를 변경하려면 괄호를 사용한다.
- 비트단위의 값을 조작하는 비트연산자를 제공한다.
- 단항 연산자들은 이항 연산자보다 우선순위가 높다.
- 파이썬은 random, math 모듈과 같은 모듈을 import하여 다양한 함수를 이용할 수 있다.

따라하며 배우는

파이썬과 데이터 과학



Questions?