

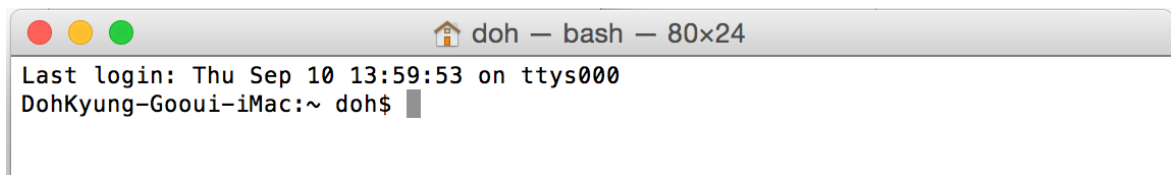
1

코딩 첫걸음

Coding, the First Step

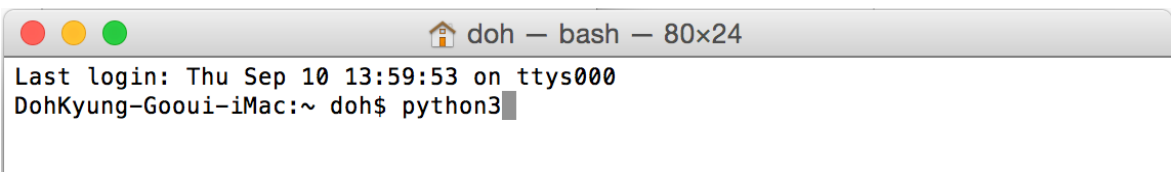
Python 실행기 준비하기

컴퓨터의 명령터미널(command-line terminal)을 띄우면 다음과 같다. 다음은 OS X의 터미널 모양이지만, Linux 등 다른 터미널도 대동소이하다.



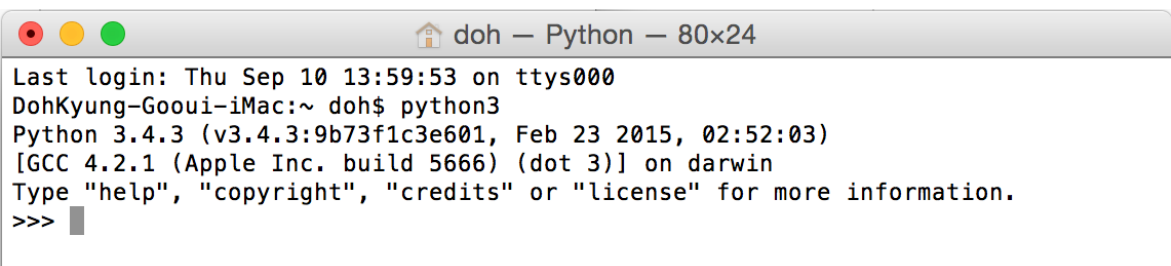
```
doh — bash — 80x24
Last login: Thu Sep 10 13:59:53 on ttys000
DohKyung-Goooui-iMac:~ doh$
```

여기서 \$ 사인은 명령을 받아들일 준비가 되어 있다는 표시이고, 그 오른쪽에 있는 회색으로 채워진 사각형은 커서(cursor)라고 하며, 키보드에서 키를 누르면 바로 그 곳에 누른 문자가 나타나면서 커서는 오른쪽으로 한칸 움직인다. Python 실행기를 켜는 명령은 python3 이므로 다음과 같이 입력하고



```
doh — bash — 80x24
Last login: Thu Sep 10 13:59:53 on ttys000
DohKyung-Goooui-iMac:~ doh$ python3
```

리턴(return)키를 누르면 다음과 같이 실행기(interpreter) 속으로 들어간다.



```
doh — Python — 80x24
Last login: Thu Sep 10 13:59:53 on ttys000
DohKyung-Goooui-iMac:~ doh$ python3
Python 3.4.3 (v3.4.3:9b73f1c3e601, Feb 23 2015, 02:52:03)
[GCC 4.2.1 (Apple Inc. build 5666) (dot 3)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

맨 아래에 있는 >>> 는 표현식 또는 명령을 기다리고 있으며 받아서 실행할 준비가 되어 있다는 표시다.

1. 키보드에서 365를 입력하고 리턴키를 눌러보자. 바로 다음 줄에 365가 나타날 것이다. 실행기는 365를 읽고 정상적으로 수의 형태임을 검사한 다음 이 수가 365임을 확인해준다.
2. 다시 키보드에서 3 + 4를 입력하고 리턴키를 눌러보자. 이번에 바로 다음 줄에 7이 나타날 것이다. 실행기는 3 + 4를 읽고 정상적인 수식의 형태임을 검사한 다음 이 수식을 계산하고 그 결과 값인 7을 보여준다.

여기서 입력한 365와 $3 + 4$ 를 표현식(expression)이라고 하며, 실행기가 답으로 보여준 결과를 값(value)이라고 한다. 즉, 실행기는 사용자가 입력한 표현식을 계산하여 그 결과 값을 보여준다. 탁상용 계산기와 다를 바 없다.

이 장에서는 Python 실행기가 처리할 수 있는 표현식을 데이터 타입 별로 공부해보자.

1. 수식 계산

Python 실행기로 수식(arithmetic expression)을 계산(evaluate) 할 수 있다.

수의 표현

수(numeral)는 정수와 부동소수점수가 있다. 십진수로 정수(integer)는 다음과 같이 표현한다.

```
0
3
365
+17
-23
```

잘 표현된 정수인지 실행기로 하나씩 확인해보자.

십진수로 부동소수점수(floating-point number)는 다음과 같이 표현한다.

```
2.5
0.0025e3
250e-2
-3.14
```

잘 표현된 부동소수점수인지 실행기로 하나씩 확인해보자.

수식

산수를 할 수 있는 이항연산자(binary operator)는 아래 표와 같이 더하기, 빼기, 곱하기, 나누기, 몫구하기, 나머지구하기, 지수승이 있다.

+	더하기
-	빼기
*	곱하기
/	나누기
//	몫구하기
%	나머지구하기
**	지수승

이항연산을 하는 수식은 관례대로 연산자(operator)가 가운데 위치하고, 피연산자(operand)가 좌우 양쪽에 위치하는 중위(infix)표기법을 쓴다. 예를 들면,

$$3 + 5$$

$$24 * 365$$

부호바꾸기(-)는 단항연산자(unary operator)로서 -3 과 같이 연산자가 피연산자 앞에 위치하는 전위(prefix)표기법을 쓴다.

- **연습문제 :** 각 연산자별로 수식을 만들어 Python 실행창에서 확인해보자. 피연산자가 정수인지 부동소수점수인지에 따라 계산 결과가 어떻게 차이나는지 수식을 만들어 관찰하면서 연산자의 의미를 모두 이해하자.

부동소수점수 오차

부동소수점수는 계산 오차가 발생할 수 있어 주의해서 다루어야 한다. 다음은 부동소수점수 0.1을 실행기로 확인하면 그대로 문제없이 0.1임을 확인해준다.

```
>>> 0.1
```

```
0.1
```

그런데 다음과 같이 0.1의 곱셈을 계산하면 결과가 0.01이 아닌 수를 계산결과로 내준다.

```
>>> 0.1 * 0.1
```

```
0.010000000000000002
```

약간의 계산 오차가 생겼다. 이러한 오차는 왜 생길까? 컴퓨터로 곱셈을 하려면 0.1을 이진수로 바꾸어야 하는데 0.1을 이진수로 정확하게 표현할 수 없기 때문이다. 정확하게 표현이 불가능하니 근사치로 표현한 다음 곱셈을 하기 때문에 그만큼 오차가 생긴 것이다.

컴퓨터에서 데이터는 모두 이진수로 바꾸어 계산한다. 부동소수점수의 소수점 이하의 수도 정수와 마찬가지로 이진수로 저장한다. 십진수 0.125는 $1/10 + 2/100 + 5/1000 = 125/1000 = 1/8 = 0/2 + 0/4 + 1/8$ 이므로 이진수 0.001로 표현가능하다. 그러나 불행하게도 이진수로 정확히 표현할 수 없는 부동소수점수가 훨씬 많다. 0.1은 수많은 사례 중 하나일 뿐이다. 소수점 아래를 근사치 이진수로 바꾸어 계산하고, 계산결과를 다시 십진수로 바꾸니 오차가 생기는 것이다. 그래서 부동소수점수는 항상 오차를 염두에 두고 프로그램을 작성해야 한다.

- **연습문제 :** 이진수로 표현가능한 부동소수점수를 십진수로 바꾸어서 어떤 수가 이진수로 정확히 표현 가능한지 알아보자.
- **연습문제 :** 이진수로 표현불가능한 부동소수점수를 다섯개 찾아서 계산 오차가 생김을 Python 실행창에서 확인해보자.

우선순위

$2 * 3 + 4$ 와 같은 수식은 어떤 연산을 먼저 하느냐에 따라 결과가 달라진다. 곱셈을 먼저하면 10이 되고, 덧셈을 먼저하면 14가 된다. 관례적으로 연산자에 우선순위(precedence)를 미리 정해두고 그 순서대로 연산한다. 연산자의 우선순위는 다음 표와 같다.

가장높음	**	
높음	-	부호바꾸기
낮음	*	
	/	
	//	
	%	
가장낮음	+	
	-	빼기

우선 순위를 바꾸고 싶으면 괄호를 사용한다. 예를 들어, $2 * 3 + 4$ 에서 덧셈을 먼저 하고 싶으면 $2 * (3 + 4)$ 와 같이 쓴다.

- **연습문제 :** 우선순위가 다른 부호가 두 개 이상 들어간 수식을 만들어 실행기로 돌려보고 예상한 대로 결과 값이 나오는지 확인해보자.

결합순서

같은 연산자끼리도 계산순서에 따라 결과가 달라질 수 있다. 예를 들어, $2 - 3 - 4$ 의 경우, 왼쪽부터 연산을 하면 결과가 -5가 되고, 오른쪽부터 연산을 하면 결과가 3이 된다. 왼쪽부터 연산을 하도록 순서를 정하면 좌결합(left associative)이라고 하고, 오른쪽부터 연산을 하도록 순서를 정하면 우결합(right associative)이라고 한다. 이항연산자 중에서 지수승(**)만 우결합이고 나머지는 모두 좌결합이다.

결합순서를 바꾸고 싶으면 괄호를 사용한다. 예를 들어, $2 - 3 - 4$ 에서 오른쪽 뺄셈을 먼저 하고 싶으면 $2 - (3 - 4)$ 와 같이 쓴다.

- **연습문제 :** 결합 순서에 따라서 계산 결과가 달라지는 연산자를 찾아서 이를 보여주는 수식을 만들어 실행창에서 확인해보자.

타입 변환

타입변환 함수를 사용하여 수의 타입을 바꿀 수 있다. int 함수로 부동소수점수를 정수로 바꿀 수 있다. 소수점 이하는 버린다.

- **연습문제 :** int(3.84)의 결과를 Python 실행창에서 확인해보자.

float 함수로는 정수를 부동소수점수로 바꿀 수 있다.

- **연습문제 :** float(3)의 결과를 Python 실행창에서 확인해보자.

3. 문자열 계산

문자열의 표현

문자열(string)은 문자를 일렬로 나열해 놓은 것이다. 문자열은 큰따옴표(") 또는 작은따옴표(') 문자를 양쪽 끝에 붙여서 표현한다. 이 때 큰따옴표로 시작한 문자열은 반드시 큰따옴표로 끝나야 하며, 작은따옴표로 시작한 문자열은 반드시 작은따옴표로 끝나야 한다.

옴표로 시작한 문자열은 반드시 작은따옴표로 끝나야 한다. 이를 문자열 표현식(string expression)이라고 하며, 문자열 표현을 제대로 했는지 Python 실행창에서 확인할 수 있다.

Python 실행창에서 다음과 같이 문자열 표현식을 만들어 확인해보자.

```
'Computer Science'
"Computer Science"
```

같은 모양의 따옴표로 둘러싸지 않거나 따옴표를 빠트리면 문자열로 인식하지 못하며, 구문오류(syntax error)가 발생했다는 오류메시지를 늘어놓으며 비정상적으로 실행을 종료한다.

Python 실행창에서 다음의 틀린 문자열 표현식으로 오류메시지가 무슨 뜻인지 해독해보자.

```
'Computer Science
"Computer Science'
```

문자열 붙이기

대표적인 문자열 연산으로 문자열 붙이기(string concatenation)가 있다. 붙이고 싶은 두 문자열 사이에 더하기(+) 연산자를 넣어 문자열 붙이기 표현식을 만든다. Python 실행창에서 다음 문자열 붙이기 표현식의 계산 결과를 확인해보자.

```
'Computer' + "Science"
"Computer" + " " + 'Science'
```

여기서 " "는 빈칸 하나로만 구성된 문자열이다.

연산자를 사용하지 않고 문자열을 그냥 일렬로 나열해도 문자열이 붙는다. Python 실행창에서 다음 문자열 붙이기 표현식의 연산 결과를 확인해보자.

```
'Computer' "Science"
"Computer" " " 'Science'
```

빈문자열

문자가 하나도 없는 문자열은 빈문자열(empty string)이라고 하며, "" 또는 ''로 표현한다. 빈 문자열은 빈칸이 하나 들어간 문자열과 다름을 주의하자. 다음 문자열을 실행창에서 하나씩 확인해보자.

```
''
""
'Computer' + "" + 'Science'
```

타입 변환

Python에서 문자열과 수를 바로 붙일 수 없다. 다음과 같이 문자열과 수의 붙이기를 시도하면 타입오류(type error)가 발생했다는 오류메시지를 늘어놓으며 비정상적으로 실행을 종료한다.

```
"World Cup " + 2014
```

위의 붙이기를 실행창에서 시도하고 오류메시지가 무슨 뜻인지 해독해보자.

정수와 부동소수점수는 str 함수를 사용하여 문자열로 바꿀 수 있다. 예를 들어, str(2014)은 "2014"를 만들어 준다.

- **연습문제** : 바로 위의 예를 str 함수를 사용하여 "World Cup 2014"를 만들어보자.

부동소수점수가 문자열로 어떻게 바뀌는지 다음을 실행창에서 각각 확인해보자.

```
str(3.14)
```

```
str(0.2e3)
```

정수로 표현된 문자열은 int 함수를 사용하여 정수로 바꿀 수 있다. 정수로 표현된 문자열이 정수로 잘 바뀌는지 다음 예를 가지고 실행창에서 확인해보자.

```
int("123")
```

```
int("3+4")
```

부동소수점으로 표현된 문자열은 float 함수를 사용하여 부동소수점으로 바꿀 수 있다. 부동소수점으로 표현된 문자열이 부동소수점으로 잘 바뀌는지 다음 예를 가지고 실행창에서 확인해보자.

```
float("3.14")
```

```
float("0.3e2")
```

부동소수점 형식을 갖춘 문자열을 int 함수를 사용하여 정수로 만들수 있나? 실행창에서 다음을 확인하면 대답할 수 있다.

```
int("3.14")
```

```
int("0.3e2")
```

정수 형식을 갖춘 문자열을 float 함수를 사용하여 부동소수점으로 만들수 있나? 실행창에서 다음을 확인하면 대답할 수 있다.

```
float("3")
```

반복 붙이기

같은 문자열을 여러 번 반복해서 붙이고 싶으면 곱하기(*) 연산자를 사용한다. <문자열> * n 은 <문자열>을 n번 연속해서 붙인다는 의미이다. 여기서 n이 0 이하의 정수이면 결과는 빈문자열이다.

다음의 결과를 실행창에서 각각 확인해보자.

```
"Pooh" * 5
```

```
"Pooh" * 0
```

```
"Pooh" * -3
```

구분문자와 특수문자

문자열의 시작과 끝을 표시하는 따옴표 문자를 구분문자(delimiter)라고 한다. 그런데 만약 구분문자를 문자열 내부에 넣고 싶으면 어떻게 해야할까? 다음과 같이 하면 구문오류로 처리한다.

```
'Halley's Comet'
```

왜? 위와 같이 그냥 넣어버리면 둘째 작은따옴표를 문자열의 끝으로 인식하고, 셋째 작은따옴표는 짝이 없는 따옴표가 되어 구문오류로 처리한다.

1. 해결책 하나 — 모양이 다른 따옴표를 구분문자로 사용한다. 즉, 큰따옴표를 넣고 싶으면 작은따옴표로 둘러싸고, 작은따옴표를 넣고 싶으면 큰따옴표로 둘러싼다. 따라서 위의 예는 다음과 같이 하면 원하는 대로 된다.

```
"Halley's Comet"
```

2. 해결책 둘 — 다음과 같이 역슬래시(backslash, \) 문자를 따옴표 앞에 붙여 구분문자를 탈바꿈(escape)한다. 탈바꿈하면 따옴표 문자를 구분문자로 인식하지 않는다.

```
'Halley\'s Comet'
```

- **연습문제** : 위의 문자열을 모두 Python 실행창에서 확인해보자.

줄바꿈(newline)이나 탭(tab)도 특수문자를 사용하여 문자열에 포함할 수 있다. 줄바꿈 문자는 `\n` 으로, 탭 문자는 `\t` 로 표현한다. 역슬래시 문자 자체를 문자열에 포함하고 싶으면 이를 탈바꿈하여 `\\` 로 표현한다.

4. 논리식 계산

Python 실행기는 논리식(Boolean expression)을 계산(evaluate)하여 결과 논리값을 내준다.

논리값의 표현

논리값(Boolean)은 참과 거짓으로 구성한다. 참은 `True`로 표현하고, 거짓은 `False`로 표현한다. `True`, `False` 모두, 첫자는 대문자이고 나머지는 모두 소문자이어야 함을 유의하자.

- **연습문제** : `True`와 `False`를 실행창에서 확인해보자.

논리식

많이 쓰이는 논리연산자(Boolean operator)는 논리곱(and), 논리합(or), 논리역(not)이 있다. 논리곱과 논리합은 이항연산자로 중위표기법(연산자가 가운데 위치)을 쓰고, 논리역은 단항연산자로 전위표기법(연산자가 앞에 위치)을 쓴다. 다음 논리식을 실행창에서 하나씩 확인하면서 논리연산자의 의미를 모두 이해하자.

```
True and True
True and False
False and True
False and False
True or True
True or False
False or True
False or False
not True
not False
not (False or ((not False) and True))
(not False) or ((not False) and True)
```

우선순위

논리연산자에도 우선순위가 있다.

가장높음	not
높음	and
낮음	or

다음의 논리식을 실행창에서 확인하면서 우선순위를 이해해보자.

```
not False or not False and True
```

결합순서

결합순서는 좌결합을 관례적으로 사용한다. 하지만 결합 순서는 최종 결과에 영향을 미치지 않으므로 중요하지 않다. (산수에서 덧셈, 곱셈과 성질이 같음)

계산순서

이항연산자인 논리곱과 논리합의 경우 왼쪽 피연산자를 먼저 계산한다. 만약 and 연산자의 경우 왼쪽 피연산자의 논리값이 False이면 오른쪽 피연산자를 계산하지 않고 바로 False로 결론짓는다. or 연산자의 경우 왼쪽 피연산자의 논리값이 True이면 오른쪽 피연산자를 계산하지 않고 바로 True로 결론짓는다. 즉, 계산 결과가 명확히 밝혀지면 필요없는 계산은 생략한다. 이를 단축계산(short-circuit evaluation)이라 한다. 아래 예를 실행해보면서 단축계산을 이해해보자. 여기서 loop()은 끝나지 않는 프로그램이다.

```
def loop() : loop()
True and loop()
False and loop()
True or loop()
False or loop()
```

비교논리식

비교연산은 두 값을 비교하여 같거나 다름을 판정하거나, 두 값을 비교하여 크거나 작음을 판정하는 연산으로 결과를 논리값(참 또는 거짓)으로 내준다. 따라서 비교식도 논리식이다.

Python에서 주로 많이 쓰는 비교연산자(comparison operator)는 같다(==), 다르다(!=), 크다(>), 작다(<), 크거나같다(>=), 작거나같다(<=)가 있다. 모두 이항연산자로 중위표기법(연산자가 가운데 위치)을 쓴다.

- **연습문제** : 다음 비교논리식을 하나씩 실행창에서 확인하면서 비교연산자의 의미를 모두 이해해보자.

11 == 11	True != True	"ERICA" < "ERICA"
11 != 11	True < False	3 == "three"
11 <= 11	False < True	3 >= "three"
11 < 11	True <= True	3 < "three"
12 <= 23 <= 34	False < False	True == "True"
12 <= 23 >= 14	"ERICA" == "ERICA"	False == 0
3 == 3.0	"ERICA" == "erica"	True == 1
3 == 3.1	"ERICA" != "erica"	True == 3
True == False	"ERICA" <= "Hanyang"	
False == False	"ERICA" <= "ERICA"	

5. 표현식

2,3,4절에서 배운 산수식, 문자열 표현식, 논리식을 통틀어 표현식(expression)이라고 한다. 표현식은 계산(evaluate)하면 결과로 값을 내준다. 즉, 산수식을 계산하면 수값을 결과로 내주고, 문자열 표현식을 계산하면 문자열을 결과로 내주고, 논리식을 계산하면 논리값을 결과로 내준다.

지금까지 표현식의 계산 결과는 다음과 같은 형식으로 실행창을 통해서 직접 확인할 수 있었다.

```
<표현식>
=> 결과값
```

6. 출력

표준 출력

Python 실행창에 계산 결과를 출력하여 보여주는 기능을 표준 출력(standard output)이라고 한다. 실행창에 계산결과를 프린트하고 싶으면 다음과 같은 형식으로 프린트 명령문(print command)을 사용한다.

```
print(<표현식>)
```

이 명령문의 의미는 다음과 같다.

<표현식>의 계산 결과를 실행창에 프린트하라.

두 개 이상 표현식을 다음과 같이 나란히 나열하여 모두 한줄에 프린트할 수도 있다.

```
print(<표현식>, ..., <표현식>)
```

이 명령문의 의미는 다음과 같다.

<표현식>을 순서대로 계산하여 결과를 실행창에 한 줄로 프린트하되 결과 사이에 빈칸을 하나씩 끼워라.

편집창 사용하기

Python 프로그램은 기본적으로 명령문(command)의 나열이며, 첫 명령문부터 차례로 하나씩 실행한다. 그런데 여러 줄의 명령문을 하나씩 실행창에 실행해보는 건 여러모로 불편하다. 따라서 편집창에 프로그램을 작성한 후에 (편집창의 오른쪽 상단에 있는) 실행단추를 눌러 한꺼번에 실행하면 편하다.

문자열 출력

문자열의 양쪽끝 따옴표는 문자열의 시작과 끝을 표시해주는 구분문자일 뿐이므로 프린트한 결과에는 보여주지 않는다.

다음 문자열을 편집창에 입력하고, 실행단추를 눌러 실행하여, 실행창에서 결과를 확인해보자.

```
print("스매시스타일")
print("소프트웨어융합대학" + " " + "스매시스타일!")
```

다음 문자열을 편집창에 입력하고, 실행단추를 눌러 실행하여, 실행창에서 결과를 확인해보자.

```
print("소프트웨어융합대학", "스매시스타일!")
```

다음은 Python 실행창에서 각각 실행해보고 왜 그런 결과가 나왔는지 생각해보자.

```
"Welcome to\n\tthe world of\n\tcomputer science!"
print("Welcome to\n\tthe world of\n\tcomputer science!")
```

수 출력

출력할 값이 정수이면 그대로 프린트하므로 별로 색다를 바가 없다. 출력할 값이 부동소수점수이면 오차를 보정한 결과를 프린트한다.

- **연습문제** : 지금까지 공부한 정수와 부동소수점수를 모두 프린트 명령문을 사용하여 창에 프린트 해보자
- **연습문제** : 일주일이 몇분인지 계산해서 프린트하는 프로그램을 편집창에 작성하여 실행해보자.

7. 변수와 지정문

표현식의 계산 결과 값으로 정수, 부동소수점수, 문자열, 논리값이 있음을 알았다. 이러한 표현식의 계산 값은 바로 보여주고 말기 때문에, 나중에 쓰려면 보관해두어야 한다. 컴퓨터에서 값을 보관할 수 있는 장소는 메모리이다. 메모리에 값을 보관해두고 필요할 때 찾아쓰려고, 보관한 메모리 주소에 이름을 붙여 두는데, 이 이름을 변수(variable)라고 한다. 즉, 변수는 값의 저장소에 붙어있는 이름이라고 보면된다. 표현식을 계산하여 값을 변수(엄밀히 말하면 변수가 가리키는 메모리)에 저장하는 작업을 지정(assignement)이라고 한다. 그리고 지정을 실행하는 명령문을 지정문(assignment command)이라고 한다. 지정문을 작성하는 문법 형식은 다음과 같다.

〈변수〉 = 〈표현식〉

〈표현식〉은 〈수식〉 또는 〈문자열 표현식〉 또는 〈논리식〉을 모두 대표한다.

〈변수〉는 반드시 문자(a-z, A-Z), 숫자(0-9), 아래줄(_)의 조합으로만 만들어야 하며, 숫자로 시작할 수 없다. 변수는 영어 알파벳 소문자로 시작하는게 관습이다.

지정문의 의미는 다음과 같다.

〈표현식〉을 계산한 결과 값을 〈변수〉로 지정한다.

다음 지정문을 편집창에 입력하고 실행해보자.

```
width = 3
height = width + 2
```

실행창에 어떤 결과가 나오나? 아무 것도 나타나지 않는다. 실행창에서 아무 것도 볼 수 없지만, 내부적으로 첫째 지정문의 오른쪽 〈표현식〉 3을 계산한 결과 값 3이 왼쪽 〈변수〉 width에 지정되며, 둘째 지정문의 오른쪽 〈표현식〉 width + 2를 계산한 결과 값 5가 왼쪽 〈변수〉 height에 지정된다. 첫 줄에서 지정된 width 변수가 다음 줄에서 바로 사용되었음을 확인하자.

다음의 프린트 문을 편집창에 추가하여 width와 height에 지정된 값을 실행창에서 확인해보자.

```
print(width, height)
```

변수에 지정된 값은 언제든지 불러 쓸 수 있다. 다음 문장 두 줄을 편집창에 추가하여 실행해보자.

```
area = width * height / 2.0
print(area)
```

변수는 불러쓰기 전에 반드시 지정해야 한다. 지정되지 않은 변수를 불러 쓰면 실행기가 오류메시지를 내놓으며 멈춘다. 실행창에서 다음을 입력하여 어떤 반응을 하는지 살펴보자. (모르는 이름이라고 불평하면서 비정상적으로 종료한다.)

```
pooh
```

변수는 언제나 다시 지정하여 새로운 값으로 고칠 수 있다.

편집창에서 다음을 네 줄 추가하고 다시 실행해보자.

```
height = height - 1
print(width, height)
area = width * height / 2.0
print(area)
```

지정문을 실행하여 새 값이 지정되면, 전에 지정된 값은 영원히 복구 불가능하다. 예를 들어, 위에서 height의 값이 7 이었는데 새 지정문에 의해서 6이 되었다. 전에 저장되어 있던 7 값은 지워지고 없다. height 변수의 값이 변경되었으니, 이에 따라 area 변수 값도 변경되었된다.

- **연습문제** : 변수 a와 b에 임의의 다른 값을 지정하고, 지정된 값을 교환하는 지정문을 작성해보자. (귀 톺: 교환하는데 지정문이 3개 필요하다)

중복 지정

한번에 여러 변수를 다음과 같이 지정할 수 있다. 실행창에서 한줄 씩 실행해보자.

```
x = y = z = 3 + 4
x
y
z
```

동시 지정

여러 변수에 동시에 여러 값을 저장할 수도 있다. 실행창에서 한줄 씩 실행해보자.

```
x, y = 2, 3
x
y
x, y = y, x
x
y
```

- **연습문제** : 변수 a와 b에 임의의 다른 값을 지정하고, 지정된 값을 교환하는 지정문을 이번엔 동시지정문으로 작성해보자. (귀 톺: 동시지정문으로는 1줄이면 충분하다.)

변수 이름붙이는 규칙

변수 이름은 반드시 문자(a-z, A-Z), 숫자(0-9), 아래줄(_)의 조합으로만 만들어야 하며, 숫자로 시작할 수 없다. 다음을 편집창에 입력하고 실행 단추를 눌러 실행해보자. 구문 오류가 생기면 오류를 수정하고 재시도 해보자.

```
erica13 = 2013
13erica = 2013
```

```
pythonProgramming = "easy"
python_programming = "fast"
```

변수 작명은 잘해야 한다. 변수로 지정되어 있는 값을 잘 대변해줄 수 있는 이름으로 지어야 프로그램의 가독성을 높일 수 있어서 좋다. 변수 작명에 고려해야 하는 사항을 나열해보면 다음과 같다.

- 값의 성격을 잘 대변해주는 이름을 고를 것
- 일관성을 유지할 것
- 관습을 따를 것 (일반 변수는 소문자로 시작)
- 너무 길게 만들지 말 것

값을 계산한 즉시 한 번 쓰고 마는 경우 굳이 변수를 지정할 필요가 없다. 그러나 위 예와 같이 연속적으로 쓰는 경우 변수로 지정해두고 필요한 대로 고쳐가면서 불러쓰면 유용하다.

변수 사용

다음 프로그램을 편집창에 입력한 후, 실행해보자.

```
print(5, "일을 분으로 따지면", 5*24*60, "분이다.")
print(7, "일을 분으로 따지면", 7*24*60, "분이다.")
print(28, "일을 분으로 따지면", 28*24*60, "분이다.")
```

여러번 반복 사용하는 데이터는 변수로 지정해주면 더 간편해지고 가독성도 좋아진다. 같은 프로그램이지만 다음과 같이 변수를 사용하면 코드의 가독성이 훨씬 좋아지고 중복을 배제할 수 있어서 코드가 간결해진다. (사실 아직 중복이 완벽히 제거된 건 아니다.) 다음을 실행해보고 같은 결과가 나오는지 확인해보자.

```
c1, c2 = "일을 분으로 따지면", "분이다."
days = 5
minutes = days * 24 * 60
print(days, c1, minutes, c2)
days = days + 2
minutes = days * 24 * 60
print(days, c1, minutes, c2)
days = days * 4
minutes = days * 24 * 60
print(days, c1, minutes, c2)
```

8. 입력

실행창에서 사용자로부터 키보드 입력을 받는 기능을 표준 입력(standard input)이라고 한다.

입력함수 input()

입력 함수 input()은 표현식의 일종으로 다음과 같은 순서로 키보드 입력을 문자열로 만들어 내준다.

1. input() 함수를 실행하면, 실행창에 커서(cursor)가 보이며 사용자의 키보드 입력을 기다린다.
2. 사용자는 키보드로 입력을 완성한 후 리턴(return) 키를 누른다.

3. 사용자의 키보드 입력을 문자열로 내준다.

실행창에서 다음을 실행하여 무엇이든 입력을 넣고, 결과를 확인해보자.

```
input()
```

입력함수의 괄호사이(인수가 들어가는 부분)에 <표현식>을 넣어주면, 그 <표현식>을 계산한 결과를 실행창에 프린트한 후 한칸 띄우고 그 뒤에 입력커서가 위치한다.

이번엔 편집창에서 다음을 입력하고 실행하여 무엇이든 입력을 넣고, 변수 x 값을 확인해보자.

```
x = input("무엇이든 주세요. ")
print x
```

입력 값의 사용

다음은 사용자에게서 날짜의 수를 받아서 그 날짜에 해당하는 총 분수를 계산하는 프로그램이다. 프로그램을 읽고 이해한 후, 편집창에 입력하고 실행해보자. 입력받은 수는 문자열이므로 정수로 바꾸어 계산한다. 5, 7, 28 입력에 대해서 각각 전과 같은 결과를 프린트하는지 확인해보자.

```
c1, c2 = "일을 분으로 따지면", "분이다."
days = int(input("며칠?"))
hours = days * 24 * 60
print(days, c1, hours, c2)
```

9. 프로그램 작성 사례 - 동전 총액 계산하기

이제 아주 간단한 프로그램을 작성할 준비가 되었다. 프로그램은 보통 다음과 같은 차례로 만든다.

1. 문제와 입력, 출력의 정의
2. 문제를 푸는 알고리즘 설계
3. 설계한 알고리즘을 기반으로 프로그램 작성 (주석 달기)
4. 실행 검사하면서 프로그램 보수

알고리즘(algorithm)이란 문제를 풀어 해답을 얻는 절차를 말한다. 이 절차를 컴퓨터가 이해하는 언어로 작성한 것이 컴퓨터 프로그램이다. 이 절차를 Python 실행기가 이해하도록 작성해놓은 프로그램이 Python 프로그램이다. 이제 간단한 사례를 통해서 프로그램 개발과정을 경험해보자.

문제

동전 지갑에 들어있는 동전의 개수를 동전의 종류별로 입력받아 총액을 계산해주는 Python 프로그램을 작성하자.

- 입력 : 동전 종류별 동전의 개수 (동전의 종류 - 500원, 100원, 50원, 10원)
- 출력 : 동전 재산 총액 (단위: 원)

알고리즘

1. 가지고 있는 동전의 개수를 500원짜리부터 시작하여 내림 순으로 입력받는다.
2. 입력값을 가지고 동전의 총액을 계산한다. 즉, 각 동전의 가치에 개수를 곱한 다음, 모두 더한다.

3. 계산한 총액을 실행창에 보여준다.

프로그램

- 입력

```
print("동전합산 서비스에 오심을 환영합니다.")
print("음수는 입력하지 마세요.")
coin500 = int(input("500원짜리는 몇개 입니까?"))
coin100 = int(input("100원짜리는 몇개 입니까?"))
coin50 = int(input("50원짜리는 몇개 입니까?"))
coin10 = int(input("10원짜리는 몇개 입니까?"))
```

- 계산

```
total = 500 * coin500 + 100 * coin100 + 50 * coin50 + 10 * coin10
```

- 출력

```
print("\n손님의 동전은 총", total, "원 입니다.")
print("저희 서비스를 이용해주셔서 감사합니다.")
print("또 들려주세요.")
```

실행 검사

작성한 프로그램이 의도한 대로 작동하는지 검사하는 과정을 실행검사(test)라고 한다.

- 연습문제 : 위 프로그램을 편집기에서 작성하고 다양한 입력으로 여러번 실행해보자.

오류

프로그램을 잘 못 작성한 경우 오류메시지를 늘어놓으며 비정상적으로 실행을 멈출 수 있다. 프로그램 안에 있는 오류를 흔히 버그(bug)라고도 하며, 오류를 찾아서 수정하는 작업을 디버깅(debugging)이라고 한다.

1. 프로그램을 문법에 맞게 작성하지 않으면 실행도 하기 전에 오류가 발생하는데 이를 구문오류(syntax error) 또는 문법오류(grammatical error)라고 한다. 오류메시지에 구문오류임이 명시되므로 문법에 맞지 않는 부분이 어딘지 찾아서 수정해야 한다. 구문오류가 있는 다음 지정문을 실행창에서 각각 확인해보자.

```
se = 3 + 4 *
print("ERICA')
```

2. 프로그램이 문법 검사를 통과하면 프로그램 실행을 시작한다. 그러나 실행 중에 문제가 생기면 비정상적으로 실행을 멈출 수 있다. 이를 실행오류(run-time error)라고 한다.
3. 연산자와 피연산자들 사이에 타입이 맞지 않아 발생하는 실행오류를 타입오류(type error)라고 한다.

```
te = "1" + 2
```

4. 맞지 않는 값을 사용하는 경우 발생하는 실행오류를 값오류(value error)라고 한다.

```
ve = int("3.14")
```

5. 0으로 나눈 수는 존재하지 않는다. 0으로 나누면 발생하는 실행오류를 나누기0오류(Zero Division Error)라고 한다. 다음의 경우 사용자가 0을 입력하는 경우 실행오류가 발생한다.

```
zde = 55 / int(input())
```

구문오류는 상대적으로 찾기 쉬우나, 실행오류는 위에서 언급한 경우 말고도 종류가 엄청나게 많고 다양한 실행검사를 통해서도 발견하지 못할 수 있다. 오류가 전혀없는 프로그램을 만들기는 일반적으로 매우 어렵다. 특히 대규모 소프트웨어에서 오류가 없다고 보장하는 건 거의 불가능하다고들 한다. 따라서 오류를 최소화하도록 소프트웨어를 개발하기 위한 가장 근본적인 대책은 프로그래머 자신이 아주 작은 부분부터 오류없이 안전한 코딩(secure coding)을 하는 습관을 키워야 한다. 안전한 코딩에 대해서는 다음에 자세히 공부한다.

주석

코드에서 # 표시 이후 그 줄의 끝까지는 주석(comments)이라 하여 실행기가 무시한다. 따라서 코드 관리 및 가독성 증진을 위해서 주석을 이용하여 다양한 정보를 기록해두는게 좋다. 프로그램의 시작부분에는 프로그램의 이름, 간단한 설명, 입출력 명세, 작성자, 작성일, 버전, 수정일 등을 명시해두면 좋다. 코드를 이해하기 쉽게 보충 설명을 붙여두는 것도 권장한다.

다음은 위에서 작성한 프로그램에 주석을 추가한 프로그램이다. 주석만 추가하였으며 실행 결과는 동일하다. 앞으로 작성하는 모든 프로그램에 이와 같은 형식으로 주석 다는 것을 습관화하도록 하자.

```
# 동전합산 서비스
# 가지고 있는 동전의 총액을 계산해주는 프로그램
# 입력: 각 동전의 개수
# 출력: 총액
# 작성자: 도경구
# 작성일: 2014년 9월 1일 (version 0.3)

# 사용자 입력받기
print("동전합산 서비스에 오심을 환영합니다.")
print("음수는 입력하지 마세요.")
coin500 = int(input("500원짜리는 몇개 입니까?"))
coin100 = int(input("100원짜리는 몇개 입니까?"))
coin50 = int(input("50원짜리는 몇개 입니까?"))
coin10 = int(input("10원짜리는 몇개 입니까?"))

# 계산
total = 500 * coin500 + 100 * coin100 + 50 * coin50 + 10 * coin10

# 결과 출력
print("\n손님의 동전은 총", total, "원 입니다.")
print("저희 서비스를 이용해주셔서 감사합니다.")
print("또 들려주세요.")
```