

단원 1 - 1 장: 프로그래밍 사고와 파이썬 기초 소개

1 장: 컴퓨터적 사고와 기본 산술 연산

1. 사고 방식으로서의 프로그래밍

이 책의 첫 번째 목표는 여러분에게 파이썬으로 프로그래밍하는 방법을 가르치는 것이다.

하지만 프로그래밍을 배운다는 것은 새로운 사고 방식을 배우는 것이므로, 이 책의 두 번째 목표는 여러분이 컴퓨터 과학 자처럼 생각하도록 돕는 것이다.

이러한 사고 방식은 수학, 공학, 자연 과학의 가장 좋은 특징들을 결합한다.

- 수학자들처럼, 컴퓨터 과학자들은 형식 언어를 사용하여 아이디어를 표현한다 - 특히 계산을 표현함.
- 공학자들처럼, 그들은 구성 요소들을 시스템으로 조립하고 대안들 사이의 장단점을 평가하며 설계함.
- 과학자들처럼, 그들은 복잡한 시스템의 행동을 관찰하고, 가설을 세우며, 예측을 테스트한다.

우리는 프로그래밍의 가장 기본적인 요소부터 시작하여 점차 발전해 나갈 것이다.

이 장에서는 파이썬이 숫자, 문자, 단어를 어떻게 표현하는지 살펴볼 것이다.

그리고 여러분은 산술 연산을 수행하는 방법을 배우게 될 것이다.

또한 연산자, 표현식, 값, 타입과 같은 용어를 포함한 프로그래밍 어휘를 배우기 시작할 것이다.

책의 나머지 부분을 이해하고, 다른 프로그래머와 소통하며, 가상 비서를 사용하고 이해하는 데 필요할 것이다.

2. 산술 연산자(Arithmetic operators)

산술 연산자는 산술 계산을 나타내는 기호이다. 예를 들어, 덧셈 기호인 `+` 는 덧셈을 수행함.

```
In [1]: 1 30 + 12
```

```
Out[1]: 42
```

빼기 기호인 `-` 는 뺄셈을 수행하는 연산자이다.

```
In [2]: 1 43 - 1
```

```
Out[2]: 42
```

곱셈 기호인 `*` 는 곱셈을 수행함.

```
In [3]: 1 6 * 7
```

```
Out[3]: 42
```

그리고 슬래시 기호인 `/` 는 나눗셈을 수행한다:

```
In [4]: 1 84 / 2
```

```
Out[4]: 42.0
```

나눗셈의 결과가 42 가 아닌 42.0 인 것을 주목함! 이는 Python에 두 가지 유형의 숫자가 있기 때문이다:

- **정수(integers)**, 소수점이나 분수 부분이 없는 숫자를 나타낸다, 그리고

- **부동소수점 숫자(floating-point numbers)**, 정수와 소수점이 있는 숫자를 나타낸다.
두 정수를 더하거나, 빼거나, 곱하면 결과는 정수이다.
하지만 두 정수를 나누면, 결과는 부동소수점 숫자가 된다.
Python은 **정수 나눗셈**을 수행하는 또 다른 연산자 `//` 를 제공함.
정수 나눗셈의 결과는 항상 정수이다.

```
In [2]: 1 84 // 2
```

```
Out[2]: 42
```

정수 나눗셈은 항상 소수점 이하를 버리고 작은 쪽으로 근사하기 때문에(즉, '바닥'을 향해 내림) '바닥 나눗셈(floor division)'이라고도 함.

```
In [6]: 1 85 // 2
```

```
Out[6]: 42
```

마지막으로, 연산자 `**` 는 거듭제곱을 수행한다; 즉, 숫자를 특정 지수로 올림:

```
In [7]: 1 7 ** 5
```

```
Out[7]: 16807
```

다른 일부 프로그래밍 언어에서는 거듭제곱을 위해 캐럿 기호인 `^` 를 사용하지만, Python에서는 이것이 XOR 이라 불리는 비트 연산자이다. 비트 연산자에 익숙하지 않다면, 그 결과가 예상치 못할 수 있다:

```
In [8]: 1 7 ^ 2
```

```
Out[8]: 5
```

여기서는 비트 연산자를 다루지 않을 것이지만, <http://wiki.python.org/moin/BitwiseOperators> (<http://wiki.python.org/moin/BitwiseOperators>)에서 비트 연산자에 대해 읽어볼 수 있다.

^^첫 번째 섹션 끝입니다. 문제를 해결하는 시간^^

2.1. 표현식(Expressions)

연산자와 숫자의 집합을 **표현식(Expression)**이라고 한다.
표현식은 임의의 수의 연산자와 숫자를 포함할 수 있다.
예를 들어, 다음은 두 개의 연산자를 포함하는 표현식이다.

```
In [9]: 1 6 + 6 ** 2
```

```
Out[9]: 42
```

거듭제곱이 덧셈보다 먼저 일어나는 것을 주목함.

Python은 수학 수업에서 배웠을 연산 순서를 따른다: 거듭제곱은 곱셈과 나눗셈보다 먼저 일어나고, 곱셈과 나눗셈은 덧셈과 뺄셈보다 먼저 일어난다

다음 예제에서는 곱셈이 덧셈보다 먼저 일어난다.

```
In [2]: 1 12 + 5 * 6
```

```
Out[2]: 42
```

만약 덧셈이 먼저 일어나기를 원한다면, 괄호를 사용할 수 있다.

```
In [3]: 1 (12 + 5) * 6
```

```
Out[3]: 102
```

모든 표현식은 **값**을 가진다.

예를 들어, 표현식 $6 * 7$ 은 42 라는 값을 가진다.

2.2. 산술 함수(Arithmetic functions)

산술 연산자 외에도 Python은 숫자와 함께 작동하는 몇 가지 **함수**를 제공한다.

예를 들어, `round` 함수는 부동소수점 숫자를 가져와 가장 가까운 정수로 반올림한다.

```
In [4]: 1 round(42.4)
```

```
Out[4]: 42
```

```
In [5]: 1 round(42.6)
```

```
Out[5]: 43
```

`abs` 함수는 숫자의 절대값을 계산한다.

양수의 경우, 절대값은 숫자 자체임.

```
In [13]: 1 abs(42)
```

```
Out[13]: 42
```

음수의 경우, 절대값은 양수이다.

```
In [14]: 1 abs(-42)
```

```
Out[14]: 42
```

이와 같이 함수를 사용할 때, 우리는 함수를 **호출**한다고 말함.

함수를 호출하는 표현은 **함수 호출**이다.

함수를 호출할 때는 괄호가 필수이다.

괄호를 생략하면 오류 메시지가 발생한다.

참고: 다음 셀은 `%expect` 를 사용한다. 이는 이 셀의 코드가 오류를 생성할 것으로 예상한다는 의미의 Jupyter "매직 명령어"이다. 이 주제에 대한 자세한 내용은 [매직 명령어 소개 \(01 1. 매직 명령어\(주피터 노트북을 더 강력하게 만드는 비밀 무기\).ipynb#Magic-Command\)](#)를 참조하십시오.

```
In [1]: 1 import thinkpython
```

In [2]:

```
1 %%expect SyntaxError
2
3 abs 42
```

Input In [2]

```
abs 42
  ^
```

SyntaxError: invalid syntax

위 코드의 첫 번째 줄은 무시하셔도 된다; 지금 이해하는 데 필요한 정보가 포함되어 있지 않다.

두 번째 줄은 오류가 포함된 코드로, 그 아래에 캐럿(^) 기호가 있어 오류가 발견된 위치를 나타낸다.

마지막 줄은 이것이 **구문 오류(syntax error)**임을 나타내며, 이는 표현식의 구조에 문제가 있다는 것을 의미한다. 이 예시에서는 함수 호출에 괄호가 필요하다는 것이 문제이다.

괄호와 값을 모두 생략하면 어떻게 되는지 살펴보겠다.

In [3]:

```
1 abs
```

Out[3]: <function abs(x, /)>

함수 이름 자체는 값을 가지는 합법적인 표현식이다.

이것이 출력될 때, 그 값은 `abs` 가 함수임을 나타내며, 함수에 대해 추가 정보를 포함하고 있어 나중에 해설함.

^^두 번째 섹션 끝입니다. 문제를 해결하는 시간^^

3. 용어집(Glossary)

산술 연산자(arithmetic operator): + 와 * 와 같은 기호로, 덧셈이나 곱셈과 같은 산술 연산을 나타낸다.

정수(integer): 소수부나 십진수 부분이 없는 숫자를 나타내는 타입이다.

부동소수점(floating-point): 정수와 소수 부분이 있는 숫자를 나타내는 타입이다.

정수 나눗셈(integer division): 두 숫자를 나누고 정수로 내림하는 연산자 // 이다.

표현식(expression): 변수, 값, 연산자의 조합이다.

값(value): 정수, 부동소수점 숫자, 또는 문자열 -- 또는 나중에 볼 다른 종류의 값들이다.

함수(function): 어떤 유용한 연산을 수행하는 명명된 명령문 시퀀스이다. 함수는 인수를 취할 수도, 취하지 않을 수도 있으며, 결과를 생성할 수도, 생성하지 않을 수도 있다.

함수 호출(function call): 함수를 실행하는 표현식 -- 또는 표현식의 일부이다. 함수 이름 뒤에 괄호 안에 인수 목록으로 구성된다.

구문 오류(syntax error): 프로그램을 파싱할 수 없게 만들어 -- 따라서 실행할 수 없게 만드는 오류이다.

Exercises

^^세 번째 섹션 시작합니다. 문제를 연습하는 시간^^

Ask a virtual assistant

강의따라 공부하면서, 가상 비서나 챗봇을 활용하여 학습을 돕는 여러 방법이 있다.

- 챗터의 특정 주제에 대해 더 알고 싶거나 이해가 명확하지 않은 부분이 있다면, 설명을 요청할 수 있다.

- 연습 문제를 해결하는 데 어려움을 겪고 있다면, 도움을 요청할 수 있다.
각 챗터에서는 가상 비서와 함께 할 수 있는 연습 문제를 제안할 것이지만, 스스로 시도해보고 자신에게 맞는 방법을

다음은 가상 비서에게 물어볼 수 있는 몇 가지 주제이다:

- 앞서 비트 연산자에 대해 언급했지만 $7 \wedge 2$ 의 값이 5인 이유를 설명하지 않았다. "파이썬에서 비트 연산자는 무엇인가요?" 또는 " $7 \text{ XOR } 2$ 의 값은 무엇인가요?"라고 물어보세요.
- 또한 연산 순서에 대해 언급했다. 더 자세한 내용은 "파이썬에서 연산 순서는 어떻게 되나요?"라고 물어보세요.
- 부동소수점 숫자를 가장 가까운 정수로 반올림하는 데 사용한 `round` 함수는 두 번째 인수를 사용할 수 있다. "round 함수의 인수는 무엇인가요?" 또는 "원주율을 소수점 셋째 자리까지 어떻게 반올림하나요?"라고 물어보세요.
- 언급하지 않은 산술 연산자가 하나 더 있다. "파이썬에서 모듈러스 연산자는 무엇인가요?"라고 물어보세요.

궁금하시다면, 가상 비서에게 다음과 같이 물어보세요: "숫자가 0.5로 끝나는 경우, 파이썬은 올림을 하나요 아니면 내림을 하나요?"

Exercises

```
In [18]: 1 # This cell tells Jupyter to provide detailed debugging information
          2 # when a runtime error occurs. Run it before working on the exercises.
          3
          4 %xmode Verbose
```

Exception reporting mode: Verbose

대부분의 가상 비서들은 파이썬에 대해 알고 있으므로, 이러한 질문에 상당히 신뢰할 만한 답변을 제공한다. 하지만 이러한 도구들이 실수를 할 수 있다는 점을 기억해야함. 챗봇에서 코드를 받았다면, 반드시 테스트해보세요!

Exercise

숫자가 0.5 로 끝날 때 `round` 함수가 어떻게 작동하는지 궁금할 수 있다. 답은 때로는 올림하고 때로는 내림한다는 것이다. 다음 예제들을 시도해보고 어떤 규칙을 따르는지 파악해 보세요.

```
In [4]: 1 round(42.5)
```

Out[4]: 42

```
In [5]: 1 round(43.5)
```

Out[5]: 44

Exercise

새로운 함수를 배울 때는 직접 시도해보고 의도적으로 실수를 해보는 것이 좋다. 그렇게 하면 오류 메시지를 배우게 되고, 나중에 그 메시지를 다시 보았을 때 무엇을 의미하는지 알 수 있게 된다. 실수를 나중에 우연히 하는 것보다 지금 의도적으로 하는 것이 더 나은 학습 방법이다.

1. 마이너스 기호를 사용하여 -2 와 같은 음수를 만들 수 있다. 숫자 앞에 플러스 기호를 붙이면 어떻게 될까요? $2++2$ 는 어떻게 될까요?
2. $4 \ 2$ 처럼 두 값 사이에 연산자가 없으면 어떻게 될까요?
3. `round(42.5)` 와 같은 함수를 호출할 때 괄호 하나 또는 둘 다 생략하면 어떻게 될까요?

```
In [2]: 1 # Solution goes here
        2
        3 2 + + 2
```

Out[2]: 4

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1
```

```
In [ ]: 1
```

2 강: 문자열, 값 & 타입, 코드 디버깅 개념 소개

1. 문자열(String)

숫자 외에도 Python은 **문자열**이라 불리는 문자의 시퀀스를 표현할 수 있다.
문자열이라 부르는 이유는 목걸이의 구슬처럼 문자들이 함께 연결되어 있기 때문이다.
문자열을 작성하려면 직선 따옴표 안에 문자 시퀀스를 넣으면 된다.

```
In [6]: 1 'Hello'
```

Out[6]: 'Hello'

또한 큰따옴표(쌍따옴표)를 사용하는 것도 가능하다.

```
In [8]: 1 "world"
```

Out[8]: 'world'

큰따옴표는 작은따옴표(어포스트로피)를 포함하는 문자열을 쉽게 작성할 수 있게 해준다. 작은따옴표는 직선 따옴표와 같은 기호이다.

```
In [22]: 1 "it's a small pig."
```

Out[22]: "it's a small "

문자열은 또한 공백, 구두점 및 숫자를 포함할 수 있다.

```
In [9]: 1 'Well, '
```

Out[9]: 'Well, '

+ 연산자는 문자열에도 작동한다; 두 문자열을 하나의 문자열로 결합하는데, 이를 **연결(concatenation)**이라고 한다.

```
In [10]: 1 'Well, ' + "it's a small " + 'world.'
```

Out[10]: "Well, it's a small world."

* 연산자도 문자열에 작동한다; 이는 문자열의 여러 복사본을 만들어 연결한다.

```
In [25]: 1 'Spam, ' * 4
```

```
Out[25]: 'Spam, Spam, Spam, Spam, '
```

다른 산술 연산자들은 문자열에 작동하지 않다.

Python은 문자열의 길이를 계산하는 `len` 이라는 함수를 제공한다.

```
In [12]: 1 len('Spam')
```

```
Out[12]: 4
```

따라서 `len` 함수는 따옴표 사이의 문자들을 계산하지만, 따옴표 자체는 계산하지 않는다는 점에 유의하세요.

문자열을 만들 때는 반드시 직선 따옴표를 사용해야 한다.

역따옴표(백틱)는 구문 오류를 발생시킨다.

```
In [13]: 1 %%expect SyntaxError
        2
        3 `Hello`
```

```
Input In [13]
```

```
`Hello`
```

```
^
```

```
SyntaxError: invalid syntax
```

스마트 따옴표(곡선 따옴표)라고도 알려진 곱슬곱슬한 따옴표도 사용할 수 없다.

```
In [ ]: 1 %%expect SyntaxError
        2
        3 'Hello'
```

^^첫 번째 섹션 끝입니다. 문제를 해결하는 시간^^

2. 값과 타입(Values and types)

지금까지 우리는 세 종류의 값을 보았다:

- 2 는 정수(integer),
- 42.0 은 부동소수점 숫자(floating-point number), 그리고
- 'Hello' 는 문자열(string)이다.

값의 종류를 **타입(type)**이라고 한다.

모든 값은 타입을 가지고 있다 -- 또는 때로는 타입에 "속한다"고 말한다.

Python은 어떤 값의 타입을 알려주는 `type` 이라는 함수를 제공한다.

정수의 타입은 `int` 이다.

```
In [14]: 1 type(2)
```

```
Out[14]: int
```

부동소수점 숫자의 타입은 `float` 이다.

```
In [29]: 1 type(42.0)
```

```
Out[29]: float
```

문자열의 타입은 `str` 이다.

```
In [30]: 1 type('Hello, World!')
```

```
Out[30]: str
```

`int` , `float` , 그리고 `str` 타입들은 함수로도 사용될 수 있다.

예를 들어, `int` 는 부동소수점 숫자를 받아서 정수로 변환할 수 있다(항상 내림하여 반올림).

```
In [15]: 1 int(42.9)
```

```
Out[15]: 42
```

그리고 `float` 는 정수를 부동소수점 값으로 변환할 수 있다.

```
In [32]: 1 float(42)
```

```
Out[32]: 42.0
```

그리고 여기 혼란스러울 수 있는 것이 있다.

따옴표 안에 일련의 숫자를 넣으면 어떻게 될까요?

```
In [ ]: 1 '126'
```

겉으로는 숫자처럼 보이지만, 실제로는 문자열이다.

```
In [11]: 1 type('126')
```

```
Out[11]: str
```

숫자처럼 사용하려고 하면 오류가 발생할 수 있다.

```
In [34]: 1 %%expect TypeError
          2
          3 '126' / 3
```

```
-----
TypeError                                Traceback (most recent call last)
Input In [34], in <cell line: 1>()
----> 1 '126' / 3
```

```
TypeError: unsupported operand type(s) for /: 'str' and 'int'
```

이 예제는 `TypeError` 를 발생시키는데, 이는 표현식의 값들(피연산자라고 불림)이 잘못된 타입을 가지고 있다는 의미다.

오류 메시지는 `/` 연산자가 `str` 과 `int` 타입의 값들을 지원하지 않는다는 것을 나타낸다.

숫자로 이루어진 문자열이 있다면, `int` 를 사용하여 정수로 변환할 수 있다.


```
In [35]: 1 int('126') / 3
```

```
Out[35]: 42.0
```

소수점이 포함된 숫자로 이루어진 문자열이 있다면, `float` 를 사용하여 부동소수점 숫자로 변환할 수 있다.

```
In [2]: 1 float('12.6')
```

```
Out[2]: 12.6
```

큰 정수를 작성할 때, `1,000,000` 처럼 자릿수 사이에 쉼표를 사용하고 싶을 수 있다.
이것은 파이썬에서 유효한 표현식이지만, 결과는 정수가 아니다.

```
In [36]: 1 1,000,000
```

```
Out[36]: (1, 0, 0)
```

파이썬은 `1,000,000` 을 쉼표로 구분된 정수 시퀀스로 해석한다.
이런 종류의 시퀀스에 대해서는 나중에 더 자세히 배우게 될 것이다.
큰 숫자를 더 읽기 쉽게 만들기 위해 밑줄을 사용할 수 있다.

```
In [37]: 1 1_000_000
```

```
Out[37]: 1000000
```

3. 형식 언어와 자연 언어(Formal and natural languages)

자연 언어는 영어, 스페인어, 프랑스어와 같이 사람들이 말하는 언어이다. 이들은 사람에 의해 설계된 것이 아니라 자연스럽게 진화했다.

형식 언어는 특정 응용 분야를 위해 사람들이 설계한 언어이다.

예를 들어, 수학자들이 사용하는 표기법은 숫자와 기호 간의 관계를 표현하는 데 특히 유용한 형식 언어이다.
마찬가지로, 프로그래밍 언어는 계산을 표현하기 위해 설계된 형식 언어이다.

형식 언어와 자연 언어는 몇 가지 공통점이 있지만 중요한 차이점이 있다:

- 모호성: 자연 언어는 모호성으로 가득 차 있으며, 사람들은 맥락적 단서와 다른 정보를 사용하여 이를 처리한다. 형식 언어는 거의 또는 완전히 모호하지 않도록 설계되었으며, 이는 맥락에 관계없이 모든 프로그램이 정확히 하나의 의미를 갖는다는 것을 의미한다.
- 중복성: 모호성을 보완하고 오해를 줄이기 위해 자연 언어는 중복성을 사용한다. 그 결과, 자연 언어는 종종 장황하다. 형식 언어는 중복성이 적고 더 간결하다.
- 문자 그대로의 의미: 자연 언어는 관용구와 은유로 가득 차 있다. 형식 언어는 말한 그대로의 의미를 갖는다.

우리는 모두 자연 언어를 사용하며 성장하기 때문에, 때로는 형식 언어에 적응하기 어려울 수 있다.

형식 언어는 자연 언어보다 더 밀도가 높아서 읽는 데 더 많은 시간이 걸린다.

또한, 구조가 중요하므로 항상 위에서 아래로, 왼쪽에서 오른쪽으로 읽는 것이 최선이 아닐 수도 있다.

마지막으로, 세부 사항이 중요합니다. 자연 언어에서는 넘어갈 수 있는 철자와 구두점의 작은 오류가 형식 언어에서는 큰 차이를 만들 수 있다.

4. 디버깅(Debugging)

프로그래머들은 실수를 한다. 재미있게도, 프로그래밍 오류는 **버그(bug)**라고 불리며 이를 추적하는 과정은 **디버깅(debugging)**이라고 한다.

프로그래밍, 특히 디버깅은 때때로 강한 감정을 불러일으킨다. 어려운 버그와 씨름하고 있다면, 화가 나거나, 슬프거나, 당혹스러울 수 있다.

이러한 반응에 대비하는 것이 대처하는 데 도움이 될 수 있다. 한 가지 접근 방식은 컴퓨터를 속도와 정밀함 같은 특정 강점과 공감 능력의 부재 및 큰 그림을 파악하지 못하는 것과 같은 특정 약점을 가진 직원으로 생각하는 것이다.

코드의 좋은 관리자가 되는 방법: 강점을 활용하고 약점을 완화할 방법을 찾는 것이다. 그리고 효과적으로 일할 수 있는 능력을 방해하지 않으면서 문제에 대응하기 위해 감정을 사용하는 방법을 구해야함.

디버깅을 배우는 것은 좌절스러울 수 있지만, 프로그래밍을 넘어 많은 활동에 유용한 가치 있는 기술이다. 각 장의 끝에는 이 부분처럼 디버깅에 대한 제 제안이 있는 섹션이 있다. 도움이 되길 바란다!

^^두 번째 섹션 끝입니다. 문제를 해결하는 시간^^

5. 용어집(Glossary)

문자열(string): 문자의 시퀀스를 나타내는 타입.

연결(concatenation): 두 문자열을 끝과 끝으로 이어 붙이는 것.

타입(type): 값의 범주. 지금까지 본 타입은 정수(타입 `int`), 부동소수점 숫자(타입 `float`), 그리고 문자열(타입 `str`)이다.

피연산자(operand): 연산자가 작용하는 값들 중 하나.

자연 언어(natural language): 사람들이 말하며 자연스럽게 진화한 언어.

형식 언어(formal language): 수학적 아이디어나 컴퓨터 프로그램과 같은 특정 목적을 위해 사람들이 설계한 언어. 모든 프로그래밍 언어는 형식 언어이다.

버그(bug): 프로그램의 오류.

디버깅(debugging): 오류를 찾고 수정하는 과정.

^^세 번째 섹션 시작합니다. 문제를 연습하는 시간^^

Exercise

모든 표현식에는 값이 있고, 모든 값에는 유형이 있으며, `type` 함수를 사용하여 모든 값의 유형을 찾을 수 있다. 다음 표현식의 값의 유형은 무엇일까요? 각각에 대해 최선의 추측을 한 다음 `type` 을 사용하여 알아보시오.

- 765
- 2.718
- '2 pi'
- `abs(-7)`
- `abs(-7.0)`
- `abs`
- `int`
- `type`

```
In [38]: 1 type(765)
```

```
Out[38]: int
```

```
In [39]: 1 type(2.718)
```

```
Out[39]: float
```

```
In [ ]: 1
```

```
In [ ]: 1
```

Exercise

다음 질문은 산술 표현을 쓰는 연습을 할 수 있는 기회다.

1. 42분 42초에는 몇 초가 있나?
2. 10킬로미터에는 몇 마일이 있나? 힌트: 1마일에는 1.61킬로미터가 있다.
3. 10km 레이스를 42분 42초에 완주한다면, 1마일당 평균 페이스는 몇 초인가?
4. 1마일당 평균 페이스는 분과 초 단위로 몇 분인가?
5. 시속 마일 단위로 평균 속도는 얼마인가?

변수에 대해 이미 알고 있다면 이 연습에 변수를 사용할 수 있다.

모르는 경우 변수를 사용하지 않고 연습할 수 있으며, 다음 장에서 변수를 살펴보겠다.

```
In [40]: 1 # Solution goes here
          2
          3 42 * 60 + 42
```

```
Out[40]: 2562
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1 # Solution goes here
```

```
In [ ]: 1
```

```
In [ ]: 1
```