Chapter 1

프로그래밍의 정도

이 책의 목적은 컴퓨터 과학자처럼 생각하는 방법을 알려주는 것이다. 그 과정에는 수학과 공학 그리고 자연 과학의 몇 가지 대표적인 요소들을 통합한다. 수학자들처럼 컴퓨터 과학자들도 아이디어(구체적으로는 연산들)를 표현하기 위해 형식을 갖춘 언어를 사용한다. 공학자들처럼 무엇인가를 설계하고 요소들을 조립하여 시스템을 만들기도 하며 대안들을 평가하여 타협점을 찾는다. 과학자들처럼 복잡한 시스템들의 동작을 관찰하고 가설을 세워서 예측치를 실험한다.

컴퓨터 과학자에게 있어서 가장 중요한 기술은 **문제 해결**이다. 문제 해결이라는 것은 문제를 정의한 후 해답을 찾기 위해 창의적으로 생각하여 얻은 해답을 정확하고 명료하게 표현하는 것이다. 나중에 알게 되겠지만, 프로그래밍을 배우는 과정은 문제 해결 능력을 키우는 최고의 기회이다. 그래서, 이 장의 제목을 "프로그래밍의 정도"라 지었다.

어떤 수준에서는 프로그래밍을 배우게 될 것이다. 그 자체로도 매우 유용한 기술이다. 또 다른 수준에서는 프로그래밍이 수단이 될 것이다. 계속 따라오다 보면 프로그래밍이 수단이 된다는 것이 명확해질 것이다.

1.1 프로그램이란 무엇인가?

프로그램이란 명령들의 순서로서 어떤 방식으로 연산할지 정한다. 연산은 수식을 풀어내는 것이나 다항식의 해를 구하는 것처럼 수학적일 수도 있지만 문서의 어떤 글을 찾거나 교체한다거나 이미지 처리나 동영상을 재생하는 것처럼 상징적인 연산일 수도 있다.

세부적인 것은 언어마다 다르겠지만 몇 가지 기본적인 명령들은 거의 모든 언어에 나타난다:

입력(input): 키보드나 파일, 네트워크나 다른 장치로부터 데이터를 받는다.

출력(output): 데이터를 화면에 표시하거나 파일에 저장 또는 네트워크로 전송 등을 한다.

계산(math): 덧셈과 곱셉과 같은 기본적인 수학적 연산을 한다.

조건부 실행(conditional execution): 조건을 확인하여 적절한 코드를 실행한다.

반복(repetition): 대체적으로 약간의 변형을 포함하는 어떤 행동을 반복적으로 수행한다.

믿거나 말거나, 지금 열거한 내용이 거의 전부이다. 지금까지 사용했었던 모든 프로그램이 얼마나 복잡하든 이러한 명령들로 이루어져있다. 그렇기 때문에 프로그래밍이란 것은 크고 복잡한 어떤 작업을 이런 기본 명령들로 동작 가능한 단위들로 작게 자르는 일이라고 보면 된다.

1.2 Python 실행하기

Python을 시작하는데 있어서 도전거리 중의 하나는 Python과 관련 소프트웨어를 컴퓨터에 설치해야 할 수도 있다는 것이다. 현재 사용하는 운영체제와 익숙하다면, 특히 명령줄 기반의 인터페이스와 익숙하다면, Python을 설치하는데 전혀 어려움이 없을 것이다. 하지만, 초보자라면 시스템 관리와 프로그래밍을 동시에 배워야 한다는 것이 고통스러울 수 있다.

이 문제를 해결하기 위해서 Python을 브라우저에서 시작하기를 추천한다. 나중에는 Python과 익숙해졌을 즈음에 컴퓨터에 Python을 설치하는 것을 권하도록 하겠다.

Python을 실행할 수 있는 여러 웹 페이지들이 있다. 이미 자주 사용해왔던 사이트가 있다면 그것을 사용해도 된다. 만약 없었다면 PhythonAnywher라는 곳을 추천한다. 이 URL(`http://tinyurl.com/thinkpython2e`)에 시작하기 위한 상세한 정보를 설명해 놓았다.

Python에는 두 개의 버전이 있다. Python 2와 Python 3 이다. 둘은 매우 유사하여 하나를 배우면 다른 하나로 이동하는 것은 쉽다. 사실 초보자 입장에서는 별반 차이가 없다. 이 책은 Phython 3로 작성되어 있지만, Python 2에 대하여 메모를 남겨 놓기는 했다.

Pthon **인터프리터(interpreter)**는 Python 코드를 읽고 실행하는 프로그램이다. 환경에 따라 인터프리터를 아이콘을 클릭하거나 명령줄에 `phtyon`을 입력해서 시작할 수도 있다. 시작하면 다음과 같은 출력을 보게 될 것이다.

```
Python 3.4.0 (default, Jun 19 2015, 14:20:21)
[GCC 4.8.2] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

처음 세 줄은 인터프리터에 대한 정보와 사용 중인 운영체제를 나타내기 때문에 환경에 따라 다르게 나타날 수 있다. 하지만, 버전은 확인해야 한다. 이 예제에서는 `3.4.0`으로 되어 있고, 3으로 시작한다. 즉, Python 3을 실행 중이라는 것을 뜻한다. 만약 2로 시작한다면 (예상한 것처럼) Python 2를 실행중이다.

마지막 줄은 **프롬프트(prompt)**로서 인터프리터가 코드를 입력받을 준비가 되었다는 것을 뜻한다. 코드 한 줄을 쓰고 엔터를 치면 인터프리터가 결과를 보여준다.

```
>>> 1 + 1
2
```

이제 시작할 준비가 되었다. 이제부터는 Python 인터프리터를 시작하는 방법과 코드를 실행하는 방법을 안다고 가정하겠다.

1.3 최초의 프로그램

전통적으로 새로운 언어를 사용하여 작성하는 최초의 프로그램을 "Hello, World!"라고 부른다. "Hello, World!"라고 표시하는 것이 전부인 프로그램이기 때문이다. Python으로 만든 이 프로그램은 다음과 같다.

```
>>> print('Hello, World!')
```

이것은 **print 문**의 예제이다. 실제 종이 인쇄하는 것이 아니고 화면에 결과를 표시한다. 이 경우에는 결과는 다음의 단어들이다.

```
Hello, World!
```

프로그램에 사용된 따옴표가 표시될 글씨의 시작과 끝을 나타내며 결과에는 표시되지 않는다.

괄호는 `print`가 함수라는 것을 나타낸다. 함수에 대해서는 3 장에서 다룰 것이다.

Python 2에서는 `print`문은 함수가 아니라서 괄호를 사용하지 않는다는 것이 약간 다르다.

```
>>> print 'Hello, World!'
```

이러한 차이는 차차 이해하게 될 것이다. 하지만 이 정도만 알아도 시작하는데는 충분하다.

1.4 Arithmetic operators

"Hello, World" 다음으로 해 볼 것은 산수이다. Python은 덧셈과 곱셉의 연산을 나타내는 특수한 기호들을 **연산자(operators)**를 제공한다.

+와 - 그리고 * 연산자는 덧셈과 뺄셈 그리고 곱셉을 나타낸다. 다음의 예제를 살펴보자.

```
>>> 40 + 2
42
>>> 43 - 1
42
>>> 6 * 7
42
```

/ 연산자로 나눗셈을 할 수 있다.

```
>>> 84 / 2
42.0
```

이 계산의 결과가 왜 42이 아니고 42.0인지 궁금할 것이다. 그것에 대해서는 다음 절에서 설명하도록 하겠다.

마지막으로 ** 연산자는 거듭제곱을 계산할 때 사용된다. 그 수를 지수로 사용한다.

```
>>> 6**2 + 6
42
```

어떤 언어들에서는 ^가 거듭제곱 연산에 사용되기도 하지만 Python에서는 그 기호는 비트단위의 연산자인 XOR을 나타낸다. 비트단위 연산자들이 익숙하지 않다면 연산 결과에 놀라워할 것이다.

```
>>> 6 ^ 2
4
```

이 책에서는 비트단위 연산자들에 대해서는 다루지 않겠지만, 더 알고 싶다면 http://wiki.python.org/moin/BitwiseOperators에서 읽을 수 있다.

1.5 값과 형

값(value)은 프로그램이 다루는 글자와 숫자와 같은 기본적인 것 중 하나이다. 지금까지 본 것 중에는 2, 42.0 그리고 'Hello, World!'가 있다.

이 값들은 서로 다른 **형(type)**에 속한다: 2은 **정수(integer)**형, 42.0은 **부동 소수점(floating-point)**형의 숫자이다. 'Hello, World!'는 **문자열(string)**형으로 불리는데 마치 글자들이 줄에 매달린 듯하기 때문이다.

만약에 값이 어떤 형인지 알고 싶다면, 인터프리터가 알려줄 수 있다:

```
>>> type(2)
<class 'int'>
>>> type(42.0)
<class 'float'>
>>> type('Hello, World!')
<class 'str'>
```

이 결과들을 보면 "클래스(class)"라는 단어가 종류 또는 범주를 나타내는 의미로 사용되고 있음을 알 수 있다. 형이라는 것은 값의 종류이다.

놀랍지않게도, 정수는 `int` 형에 속해있고 문자열은 `str`에 속했으며 부동 소수점은 `float`에 속해있다.

그러면 `'2'`나 `'42.0'`과 같은 값들은 어떤 형을 갖을까? 숫자처럼 보이지만 따옴표 안에 있기 때문에 문자열 형에 속한다.

```
>>> type('2')
<class 'str'>
>>> type('42.0')
<class 'str'>
```

확인한 것처럼 문자열이다.

매우 큰 정수를 입력할 때 `1,000,000`처럼 숫자 사이에 쉼표를 넣고 싶은 유혹이 일 수도 있다. 이와 같은 표현은 유효한 *정수*형은 아니기는 하지만, Python에서 유효한 표현이기는 하다:

```
>>> 1,000,000
(1, 0, 0)
```

이 결과는 우리가 기대한 것이 전혀 아니다! Python은 `1,000,000`을 쉼표로 구분된 수열로 해석한다. 이와 같은 수열에 대해서는 이후에 더 자세히 살펴보도록 하겠다.

1.6 형식 언어 그리고 자연어

자연어(Natural language)는 사람들이 말하는 영어, 스페인어와 프랑스어와 같은 언어이다. 사람들에 의해 설계된 것은 아니지만(사람들이 언어에 대해 체계를 정립하기는하였지만) 언어는 자연적으로 발달되었다.

형식 언어(Formal language)는 특정 응용처를 위해서 사람들이 설계한 것이다. 예를 들어 수학자들이 사용하는 표기법이 형식 언어의 일종으로 숫자와 기호들의 관계를 나타내기에 좋다. 화학자들은 형식언어를 사용하여 분자들의 화학 구조를 나타낸다. 그리고 가장 중요한 것은:

> **프로그래밍 언어는 연산을 표현하기 위해 설계된 형식 언어이다.**

형식 언어는 문장의 구조를 결정하는 엄격한 **문법(syntax)** 규칙을 갖고 있다. 예를 들어 수학에서 $3+3=6$이라는 문장은 올바른 문법으로 작성되었지만 $3+=3\$6$은 그렇지 않다. 화학에서는 H_2O는 문법적으로 올바른 식이지만 ${}_2Zz$은 아니다.

문법 규칙은 **토큰(token)**과 구조의 두 형태를 갖고 있다. 토큰은 언어의 기본적인 구성 요소로서 단어나 숫자 그리고 화학에서의 원소들과 같은 것이다. $3+=3\$6$라는 식에서의 문제점은 $ 이 (최소한 내가 알기로는) 수학에서 유효한 토큰은 아니라는 것이다. 유사하게 ${}_2Zz$도 유효하지 않다. Zz를 약어로 갖는 원소가 없기 때문이다.

두 번째 종류의 문법 규칙은 토큰들을 결합하는 것과 관련이 있다. $3+=3$는 규칙에 어긋난다. 이 식에서 $+$와 $=$이 유효한 토큰이기는 하지만 연이어서 두 개를 사용할 수가 없다. 화학 식에서도 마찬가지로 아래 첨자는 이름 뒤에 나타나지 그 앞에 나타나지 않는다.

이 문장은 @ 구조적으로 잘 정의되었z만 유효하지 않은 t*ken이 사용되었다. 이 문장의 토큰은 모두 유효하지만 갖고 있다 구조를 잘못된.

모국어로 문장을 읽거나 형식 언어로 문장을 읽을 때에는 그 구조를 파악해야 한다(자연어를 파악하는 과정은 무의식적으로 일어나기는 한다). 프로그래밍에서 구조를 파악하는 과정을 **파싱(parsing)**이라고 부른다.

형식 언어와 자연어 사이에는 토큰과 구조 그리고 문법과 같은 많은 공통적 기능들이 있지만 차이점도 분명하다:

모호성(ambiguity): 자연어는 모호하기 때문에 맥락과 다른 정보들을 통해 이해를 한다. 형식 언어의 경우 거의 또는 완전하게 그 모호성의 없애도록 설계되었기 때문에 문맥과 상관없이 유일한 의미를 갖고 있다.

중복성(redundancy): 자연어가 갖고 있는 모호성을 없애고 오해를 줄이기 위해서 엄청난 중복성을 허용하고 있다. 그 결과로 인해 장황해지기 쉽다. 형식 언어는 훨씬 간결하다.

직역성(literalness): 자연어는 숙어와 은유적 표현으로 넘쳐난다. "The penny doppred(돈 떨어졌다)"라고 말을 하면 실제 뜻은 돈에 관한 것도 아니고 실제로 무언가가 떨어지는 것도 아니다(이 숙어는 뜻은 '이제야 알아 들었다'는 것이다). 형식 언어에서는 말하는 그대로가 의도한 뜻한다.

자라면서 자연어를 구사하였기 때문에 형식 언어에 익숙해지는데 많은 시간이 걸리곤 한다. 자연어와 형식 언어간의 차이는 시와 산문 간의 차이와도 유사하다.

시: 사용되는 단어의 소리도 그 단어의 의미만큼 중요하며, 시 전체 봐야만 어떤 감정적 반응을 하거나 또는 영향을 느낄 수가 있다. 애매모호한 표현이 흔할 뿐만 아니라 의도된 것일 때도 있다.

산문: 단어가 갖고 있는 의미가 좀 더 중요하다. 그렇기 때문에 구조에 의미가 더 많이 담겨 있다. 산문은 시보다는 분석하기에 더 쉽기는 하지만 모호할 때가 많다.

프로그램: 컴퓨터 프로그램의 의미는 모호하지 않아며 문자적으로 이해할 수 있어서 구조와 토큰을 분석하는 것만으로 모든 것을 이해할 수 있다.

형식 언어는 자연어보다 좀 더 난해해서 읽는데 오래 걸린다. 또한, 구조가 중요하기 때문에 위에서 아래로 왼쪽에서 오른쪽으로 읽는 것이 늘 최선의 방식은 아니다. 대신에 프로그램을 상상의 공간에서 분석하고 토큰은 구별해 내고 구조를 해석하는 방법을 익혀야 한다. 마지막으로 세부사항이 중요하다. 철자나 구두점의 표기를 잘못하는 것과 같은 작은 실수를 자연어에서는 지나쳐도 문제가 되지 않았겠지만 형식 언어에서는 큰 차이를 만들어 낸다.

1.7 디버깅

프로그래머들은 실수를 한다. 황당하게도 프로그래밍 오류를 **버그(bug)**라고 부르고 그것들을 해결하는 과정을 **디버깅(debugging)**이라고 부른다.

프로그래밍과 특히 디버깅 때문에 감정이 격해질 수가 있다. 매우 어려운 버그로 고생을 하고 있다면 화가나거나 무기력해지기도 또는 당혹스러워질 수도 있다.

사람들이 컴퓨터를 대할 때 아주 자연스럽게 컴퓨터가 마치 사람인양 대한다는 결과들이 있다. 잘 동작한다면 한 팀원으로 여기기도 하고 외고집스럽고 무례할 정도로 말을 안듯는다면 그런 사람을 대하는 것과 동일하게 대한다고 한다(Reeves와 Nass, *The Media Equation: How People Treat Computers, Television, and New Media Like Real People and Places*).

이와 같은 경우에 대응하는 방법들을 미리 생각해 놓는 것이 이후에 그런 일들을 당했을 때 대처하기가 쉬워진다. 한 가지 방법은 컴퓨터를 마치 일정한 능력을, 예를 들어 속도와 정확도, 갖고 있지만 공감 능력과 전체를 보는 시각이 전혀 없는 직원이라고 생각하면 된다.

당신이 할 일은 좋은 관리자가 되는 것이다: 직원의 장점을 파악하고 단점을 완화시켜주는 것이다. 그 후에 할 일은 문제 해결 과정에서 말 안듣는 컴퓨터와 감정적 씨름을 하는 대신 작업을 효율적으로 처리할 수 있는 방법들을 찾아야 한다.

디버깅 방법을 배우는 것은 좌절스러운 경험이 될 수 있겠지만, 이 기술은 프로그래밍 외에도 여러 방면에서 적용할 수 있는 매우 가치 있는 기술이기도 하다. 각 장의 끝에 있는 절에는 디버깅에 대한 조언이 적혀 있다. 도움이 되길 바란다!

1.8 용어 해설

문제 해결(Problem solving): 문제를 정의하고 해법을 찾고 표현하는 과정.

고수준 언어(high-level language): Python과 같은 언어로 사람이 읽고 쓰기 쉽게 설계된 언어.

저수준 언어(low-level language): 컴퓨터가 실행시키기 쉽게 설계된 프로그램 언어로서 "기계어" 또는 "어셈블리어"라고도 불림.

이식성(portatiblity): 한 종류의 컴퓨터 이상에서 프로그램이 실행 가능한 성질.

해석 프로그램(인터프리터/interpreter): 프로그램을 읽고 실행시키는 프로그램.

프롬프트(prompt): 사용자로부터 입력을 받을 준비가 되었다는 것을 알려주기 위한 인터프리터가 표시하는 글자들.

프로그램(program): 연산을 지정해놓은 명령어들의 집합.

print 문(print statment): Python 인터프리터가 화면에 어떤 값을 표시하도록 하는 명령어.

연산자(operator): 덧셈, 곱셈, 또는 문자열 연결과 같은 간단한 연산을 뜻하는 특수 기호

값(value): 프로그램이 조작하는 숫자나 문자열 데이터의 기본 단위 중 하나

형(type): 값들의 분류. 정수(`int`형), 부동-소수점 숫자(`float`형)과 문자열(`str`형)이 우리가 지금까지 살펴본 데이터의 형이다.

정수(integer): 자연수를 나타내는 데이터 형이다.

부동 소수점(floating-point): 분수 부분을 표현하는 숫자들.

문자열(string): 연속된 글자들을 표현하는 데이터 형.

자연어(natural language): 사람들이 말하는 자연적으로 발전한 모든 언어.

형식 언어(formal language): 수학적 개념을 표현하기 위해서나 컴퓨터 프로그램을 표현하기 위한 언어처럼 사람들이 특정 목적을 갖고 설계한 모든 언어. 모든 프로그래밍 언어는 형식 언어이다.

토큰(token): 자연어의 단어와 대응되는 개념으로 프로그램의 문법적 구조를 이루는 기본적인 요소

문법(syntax): 프로그램의 구조를 결정하는 규칙

구문 해석(파싱, parse): 프로그램을 조사하여 문법적 구조를 분석하는 것

버그(bug): 프로그램에 있는 오류

디버깅(debugging): 버그를 찾아 고치는 과정

1.9 연습 문제

문제 1.1. *이 책을 컴퓨터 앞에서 읽는 것을 권한다. 예제들을 만날 때마다 시도 해볼 수 있기 때문이다.*

새로운 기능을 실험할 때에는 실수를 많이 할 수록 좋다. 예를 들어 "Hello, world!" 프로그램에서 둘 중의 하나의 따옴표를 입력하지 않으면 어떤 일이 생길까? 둘 다 입력하지 않으면 어떻게 될까? `print`*를 잘못 입력하면 어떻게 될까?*

이런 류의 실험은 읽은 내용을 오랫동안 기억에 남게 할뿐만 아니라 각 오류 메시지가 어떤 의미를 갖는지 알 수 있기 때문에 프로그래밍할 때에도 도움이 되나. 지금 의도적으로 실수를 만들어 내는 것이 나중에 우연히 실수를 범하는 것보다 낫다.

1. *`print`문에서 괄호 중 하나 또는 둘 다 입력하지 않으면 어떻게 될까?*
2. *문자열을 출력한다고 했을 때 따옴표 중 하나 또는 둘 다 입력하지 않으면 어떻게 될까?*
3. *음수를 표현하기 위해* `-2`*처럼 빼기 부호를 사용할 수 있다. 예를 들어* `2++2`*처럼 더하기 부호를 숫자 앞에 넣은 경우에는 어떻게 될까?*
4. *수학에서는* `02`*처럼 숫자 앞에 있는 0들이 있어도 괜찮다. Python에서 동일한 방식으로 입력한다면 어떻게 될까?*
5. *두 값을 입력할 때에 사이에 연산자가 없는 경우는 어떻게 될까?*

문제 1.2. *Python 인터프리터를 실행시키고 계산기처럼 사용해보자.*

1. *42분 42초는 모두 몇 초일까?*
2. *10 킬로미터는 몇 마일일까? 참고로 1.61 킬로미터가 1 마일이다.*
3. *10 킬로미터를 42분 42초만에 달렸다면 평균 보폭언 얼마나 될까(마일당 시간을 분과 초로 표시)? 평균 속도는 시간당 마일로 표기하면 몇인가?*