2장

프로그래밍 언어를 배우는 기본 원리

2장 프로그래밍 언어를 배우는 기본 원리

2.1 프로그래밍이라는 기술

2.1 프로그래밍이라는 기술

2.1. 프로그래밍이라는 기술

- 프로그래밍을 시작하는 지금, 가능한 한 많이 코드를 작성해 보라.
- 코드 에디터를 열고 여러분이 발견한 코드를 다 입력해 보라. 코드를 복사해서 붙여 넣지 말고 직접 하나하나 작성해 보라.
- 지금 이 시점에서 여러분의 목표는 프로그래밍을 제2의 천성으로 만드는 것이지 빨리 프로그램을 작성하는 것이 아니다.

2.2 빵 굽기와 프로그래밍의 유사성

2.2.1. ‘빵 하나 굽기’라는 목표 이해하기

‘빵 하나 굽기’라는 목표를 이해하기 위한 질문

- 빵의 크기는?
- 단순한 식사용 빵인가, 다른 향이나 재료가 들어간 빵인가? 꼭 사용해야 하거나 사용하면
- 안 되는 재료가 있나? 없는 재료는 없는가?
- 어떤 도구가 필요한가? 내가 가지고 있는 도구를 사용해야 하나, 작업을 의뢰한 쪽에서 도구를 제공하나?
- 시간 제한이 있는가?
- 사용하거나 찾아볼 수 있는 레시피가 있는가? 아니면 직접 레시피를 고안해야 하나?

목표를 중간에 다시 시작하는 번거로움을 피하려면 이런 내용을 자세히 알아야 한다.

빵을 구울 때 여러 재료 조합을 직접 시도하기보다는 가지고 있는 재료로 만들 수 있는 레시피중에서 가장 단순한 것을 선택한다.

또는 아무 것도 들어가지 않은 작은 빵을 하나 굽거나, 제빵기로 시간을 절약할 수도 있다.

2.2.2. 레시피 찾기

레시피의 예

- 단계별 작업 내용과 순서
- 각 재료의 정확한 양
- 어떤 작업을 언제 몇 번 반복할지에 대한 지침
- 특정 재료를 대체할 수 있는 재료 및 대체 방법
- 만들어진 요리를 그릇에 담아 마무리하는 방법과 사람들에게 제공하는 방법

양	재료
1/4 온스	활성 드라이이스트
1 테이블스푼	소금
2 테이블스푼	버터(유채씨 기름으로 대체 가능)
3 테이블스푼	설탕
2-1/4 컵	따뜻한 물
6-1/2 컵	중력분

1. 큰 그릇에 따뜻한 물을 붓고 이스트를 푼다.
   설탕, 소금, 버터(기름), 중력분 3컵을 넣고 부드러워질 때까지 젓는다.
2. 반죽이 부드러워질 때까지 남은 밀가루를 휘젓는다.
3. 밀가루를 뿌린 반죽판에 반죽을 놓는다.
4. 겉이 부드러워지고 반죽에 탄력이 생길 때까지 치댄다.
5. 기름을 바른 그릇에 담고 반죽을 한번 돌려서 전체에 기름을 묻힌다.
6. 반죽이 든 그릇을 덮고 따뜻한 곳에 두어 부피가 2배 될 때까지
   30분~1시간 숙성시킨다.
7. 반죽을 세게 내려쳐 이스트가 부풀면서 생긴 기포를 없애고,
   밀가루를 살짝 뿌린 판에 반죽을 놓는다.
8. 반죽을 두 덩어리로 나눈다. 각각 모양을 잡는다.
   기름을 바른 가로 25cm 세로 13cm 베이킹 판 두 개에 각각 반죽을 넣는다.
9. 반죽을 덮고 부피가 2배 될 때까지 30~45분간 반죽을 재운다.
10. 190도에서 30분간(또는 껍질이 황금색을 띤 갈색이 될 때까지) 굽는다.
11. 베이킹 판에서 빵을 꺼내서 철사 선반에 올려두고 식힌다.
12. 잘라서 맛있게 먹는다!

그림 2-1
빵 레시피 예

2.2.3. 흐름도로 레시피를 시각화하기

- 레시피는 각 단계를 말로 설명한다.
- 프로그래머가 되는 방법을 이해하려면 레시피를 머릿속에서 흐름도로 시각화(데이터나 문장을 이해하기 쉬운 도표나 그림으로 표현하는 일)할 수 있어야 한다.

2.2.3. 흐름도로 레시피를 시각화하기

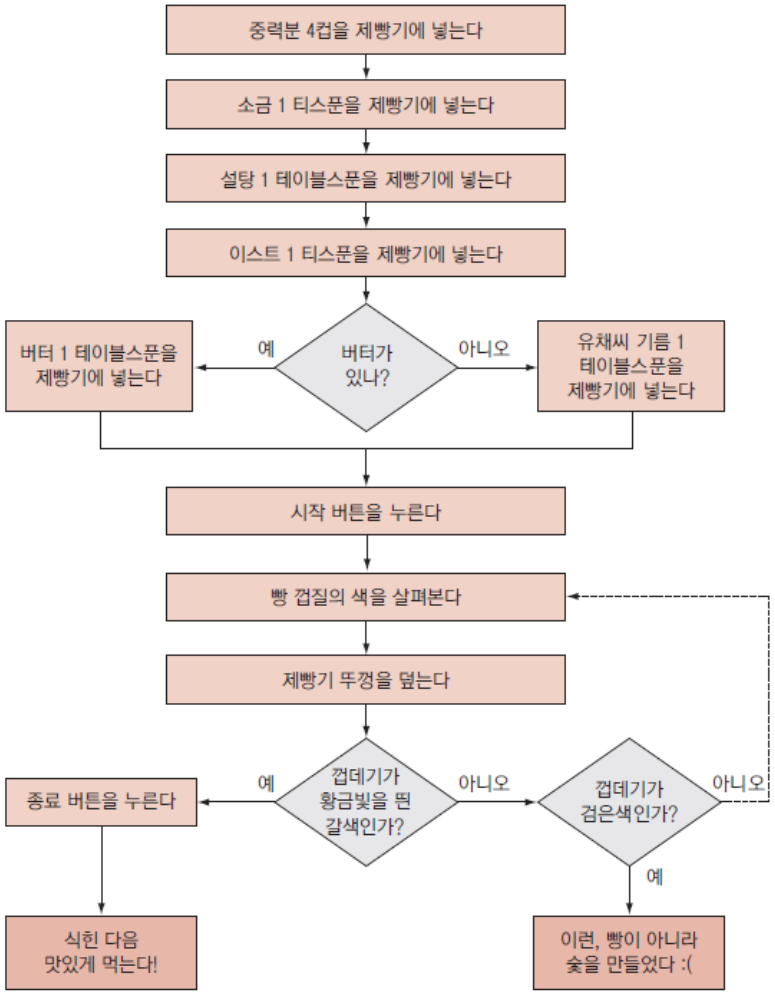

그림 2-2

빵을 굽는 간단한 레시피의 흐름도.
직사각형은 행동을, 다이아몬드는 결정해야 하는 부분을 의미한다.

더 위에 있는 이전 단계로 화살표가 되돌아가면 그 단계들을 반복하라는 뜻이다. 화살표를 따라가면서 만들어지는 다양한 경로가 이 레시피를 완성하는 여러 가지 구현 방법이다

2.2.4. 기존 레시피를 사용할 것인가 새로운 레시피를 만들 것인가?

- 제빵 예제를 통해 빵을 굽는 일에는 단순히 레시피를 따르는 것 외에 더 많은 부분이 있다는 걸 알았을 것이다.
- 먼저 구워야 할 빵이 무엇인지 이해해야 한다.
- 그 다음 따라할 만한 기존 레시피가있는지 알아보고 레시피를 결정해야 한다.
- 그런 레시피가 없다면 스스로 레시피를 고안해내고 요구 사항에 맞는 최종 결과가 만들어질 때까지 여러 번 반복해서 구워 봐야 한다.

2.3 생각하고 코딩하고 테스트하고 디버깅하기를 반복하기

2.3. 생각하고 코딩하고 테스트하고 디버깅하기를 반복하기

- 프로그램의 복잡도와 관계없이 모든 문제를 체계적이고 구조적으로 접근하는 것이 중요하다.
- 프로그램이 문제의 요구 사항을 만족시킬 때까지 '생각-코딩-테스트-디버깅-반복' 패러다임을 반복해서 사용할 것을 권한다.

2.3. 생각하고 코딩하고 테스트하고 디버깅하기를 반복하기

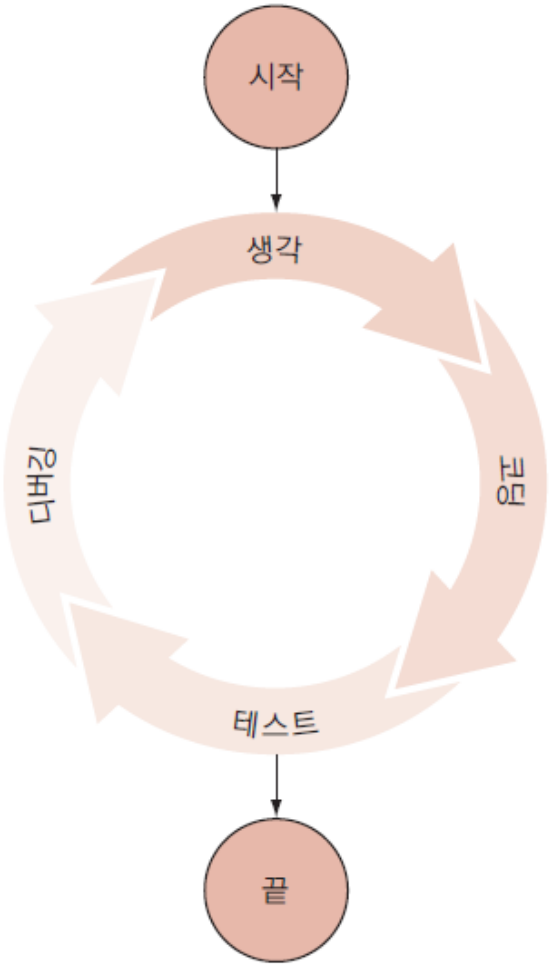

그림 2-3

이 방법은 프로그래밍으로 문제를 해결하는 이상적인 접근 방법이다. 코드를 작성하기 전에 문제를 이해하라. 그 후 작성한 코드를 테스트하고, 필요에 따라 디버깅하라. 코드가 모든 테스트를 만족할 때까지 이 과정을 반복하라

2.3. 생각하고 코딩하고 테스트하고 디버깅하기를 반복하기

생각 단계

- 여러분이 만들어 달라고 요청 받은 빵이 어떤 종류의 빵인지 확실히 이해하는 과정이다.
- 요청 받은 문제를 생각해 보고 그 문제에 들어맞는 기존 레시피가 있는지, 아니면 직접 레시피를 만들어야 할지 결정하라.
- 이런 레시피를 프로그래밍에서는 알고리즘(algorithm)이라고 부른다.

코딩 단계

- 손에 직접 반죽을 묻혀가면서 여러 재료를 조합하고, 재료를 대체하고, 반복(껍데기색을 5분마다 체크하는 것처럼)하여 시도하는 것이다.
- 프로그래밍에서는 코드로 알고리즘을 구현(implementation)한다

테스트 단계

- 요청 받은 제품과 여러분이 실제로 만든 최종 제품이 일치하는지 결정하는 것이다.
- 프로그래밍에서는 여러 다른 입력을 넣어 프로그램을 실행하고 프로그램의 실제 출력이 요구 사항에 따른 예상 출력과 일치하는지 살펴봄으로써 테스트를 진행한다.

디버깅 단계

- 레시피를 약간씩 바꾸는 것이다. 프로그래밍에서는 디버깅(debugging)으로 프로그램에게 잘못된 동작을 시키는 코드가 몇 번째 줄인지알아낸다.
- 좋은 디버깅 방식을 배워 두지 않으면 이 과정이 매우 힘들 수 있다.

2.3.1. 목표 이해하기

- 처음부터 제대로 된 코드를 작성하는 경우는 별로 없다.
- 먼저 주어진 문제에 대해 생각해야 한다.
- 문제에 대해 생각하는 단계부터 시작하면 프로그래밍 사이클을 도는 횟수를 최소화할 수 있다.
- 문제가 어려울수록 문제를 더 간단하고 작은 단계로 나누는 것이 중요하다. 우선 각각의 작은 문제를 해결하는 데 집중하자.

2.3.1. 목표 이해하기

문제가 주어지면 스스로에게 다음과 같은 질문을 던져보자.

- 이 프로그램의 목표는 무엇인가? 문장으로 정리하라. 예)원의 넓이를 구하라
- 사용자와 상호작용해야 하는가? 예)사용자가 숫자 값을 입력한다, 반지름이 사용자가 입력한 값인 원의 넓이를 구한다
- 사용자 입력 유형은 무엇인가? 예)사용자는 원의 반지름에 해당하는 숫자를 입력한다
- 사용자가 프로그램에게 기대하는 출력 형태는 무엇인가? 예) '12.57'이라고 숫자만 간단히표시하거나 '반지름이 2인 원의 넓이는 12.57입니다'처럼 자세히 표시할 수 있다. 또는 그림으로 결과를 보여줄 수도 있다.

문제를 다음 두 가지 방법으로 표현하여 생각을 정리해 보기를 권한다.

- 문제를 시각화한다.
- 입력 예제를 몇 가지 적고, 이에 어떤 출력을 예상하는지 적는다.

2.3.2. 목표 시각화하기

» 프로그램으로 해결할 목표가 있다면 그 목표를 속이 안 보이는 블랙박스라고 생각하라. 처음에는구현(implementation) 방법에 대해 걱정하지 마라.

- 구현 : 어떤 목표를 해결하기 위해 코드를 어떻게 작성하느냐를 뜻한다.

» 세세한 구현에 신경 쓰는 대신 다음 질문에 집중하라.

- 사용자와 상호작용이 필요한가? 프로그램에 입력이 필요한가?
- 프로그램이 표시해야 하는 출력이 있나?
- 프로그램 내부에서 계산만으로 모든 처리가 가능한가?

2.3.2. 목표 시각화하기

» 프로그램과 사용자 사이의 상호작용을 보여 주는 다이어그램을 그리면 도움이 된다.

- 빵 굽기 예제를 다시 생각해 보자.

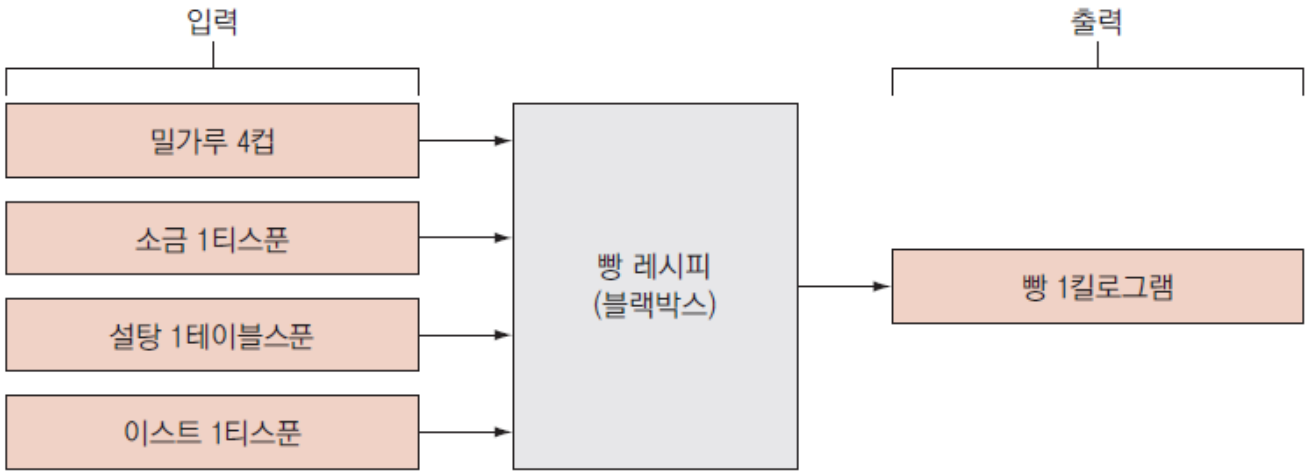

그림 2-4
주어진 재료를 사용해 빵을 굽는 과정을 블랙박스로 시각화

2.3.3. 의사 코드 작성하기

의사코드 (pseudocode)

- 종이나 코드 에디터 위에 일상 언어와 프로그래밍을 혼합해 작성한 것.
- 의사 코드를 사용하면 프로그램의 구조, 즉 사용자 입력을 받는 지점, 출력하는 지점, 몇 가지 단계를 반복 실행하는 지점 등 여러 부분을 제대로 알아볼 수 있다.

원의 넓이를 계산하는 문제를 의사 코드로 작성하면

- 사용자에게 반지름 값을 입력 받는다.
- 2. 공식을 적용한다.
- 3. 결과를 보여준다.
- 4. 사용자가 멈추라고 할 때까지 1~3단계를 반복한다.

프로그래밍으로 어떤 목표를 달성하는 방법은 수없이 많다. 한 경로를 시도해 보다가 더 이상 진행을 못하고 막혀도 괜찮다.

그 실패를 통해 구체적인 경우에 적용하기 어려운 프로그래밍 방법이어떤 것인지 더 잘 이해할 수 있다.

시간이 지나고 경험이 늘면 어떤 경우에 어떤 개념이 더 바람직한지에 대한 직관을 얻게 된다.

2.3.3. 의사 코드 작성하기

- 프로그래밍으로 어떤 목표를 달성하는 방법은 수없이 많다. 한 경로를 시도해 보다가 더 이상 진행을 못하고 막혀도 괜찮다.
- 그 실패를 통해 구체적인 경우에 적용하기 어려운 프로그래밍 방법이어떤 것인지 더 잘 이해할 수 있다.
- 시간이 지나고 경험이 늘면 어떤 경우에 어떤 개념이 더 바람직한지에 대한 직관을 얻게 된다.

2.4 읽기 쉬운 코드 작성하기

2.4.1. 의미 있고 서술적인 이름 붙이기


```
a = 3.14
b = 2.2
c = a * b * b
```

이 코드를 아래와 같이 작성했다면

```
pi = 3.14
radius = 2.2
# 원의 넓이를 계산하는 공식을 사용한다
circle_area = pi * radius * radius
```

위 코드는 반지름이 2.2인 원의 넓이를 계산한다.

*실제로는 원주율의 근삿값을 사용했으므로 정확한 계산이라기보다 추산이 맞다

- 수학과 마찬가지로 프로그래밍 언어에서는 데이터를 저장하기 위해 변수(variable)를 쓴다.
- 코드를 읽기쉽게 작성하는 데는 프로그래밍할 때 변수에 의미가 있고 서술적인(설명을 해주는) 이름을 붙이는 것이 가장 중요하다.
- 코드에서 pi는 변수 이름이고 3.14라는 값 대신 그 변수를 사용한다.
- 마찬가지로 radius와 cicle_area도 변수 이름이다.

2.4.2. 코드에 주석 달기

#으로 시작하는 줄을 주석(comment)이라고 한다.

- 파이썬에서는 주석을 # 문자로 시작하지만 다른 언어에서는 다른 특수 문자로 시작하기도 한다.
- 주석은 프로그램을 실행했을 때 작동하지 않으며, 코드에서 중요한 부분을 설명할 때사용한다.
- 주석은 왜 이렇게 코드를 작성했는지, 다른 사람과 본인이 이해하기 쉽게 설명해준다. 코드가 구현하고 있는 내용을 그냥 말로 바꾼 것이어서는 안된다.
- 주석은 특정 코드를 구현한, 감춰진 큰 아이디어를 설명해 줘야 한다. 그러면 코드를 읽는 사람은 큰 그림을 이해한 뒤 코드를 한줄한줄 살펴보면서 여러분이 어떤 계산을 하는지 구체적으로 살펴볼 수 있을 것이다.

2.5 요약

2.5. 요약

- 좋은 프로그래머라면 '생각-코딩-테스트-디버깅-반복'이라는 사이클을 반드시 따라야 한다.
- 주어진 문제를 생각하면서 요청 받은 내용을 이해해야 한다.
- 코딩을 시작하기 전에 문제 설명에 따라 입력과 출력이 무엇인지 찾아내라.
- 문제 설명이 목표를 해결하기 위해 취해야 할 일련의 단계를 표현하지 않을 수도 있다. 레시피처럼 목표를 달성하기 위한 일련의 단계 형태로 주어질 수도 있다.
- 다른 사람이 읽을 것을 염두에 두고 코드를 작성해야 한다. 이름을 붙일 때는 의미 있고 서술적인 이름을 사용해야 하며, 문제와 해법을 작성한 코드를 말로 설명해주는 주석을 작성해야 한다.