혼자 공부하는 첫 프로그래밍

with 파이썬

01-1 프로그래밍 알아보기

1. ②

해설

② 스마트폰도 컴퓨터이기 때문에 스마트폰에서 실행되는 카카오톡도 컴퓨터 프로그램입니다.

2. ②, ③

해설

소스 코드는 컴파일러 혹은 인터프리터를 통해 기계어로 번역됩니다.

01-2 "Hello, World!" 프로그램 만들기

1. ④

해설

④ 클라우드 파이썬 인터프리터는 웹 브라우저에서 코드 작성 및 실행을 모두 할 수 있으므로 별도의 파이썬 코드 편집기가 필요 없습니다.

2. ①

해설

② 파이썬 문법은 배우기 쉽다고 알려져 있습니다.

③ 파이썬의 실행 속도는 아주 빠르며, 기계 학습(머신러닝)과 같이 대용량의 수리 계산이 필요한 경우 수리 계산은 C 언어로 작성하고 나머지 부분을 파이썬으로 작성해서 상대적으로 느린 속도를 충분히 보완할 수 있어서 교육용뿐만 아니라 산업 전반에서 널리 사용됩니다.

④ 파이썬은 다른 프로그래밍 언어로 작성된 코드를 쉽게 가져가 사용할 수 있습니다.

3. ①

해설

① 파이썬 소스 코드의 확장자는 'py'입니다.

4. (3)

해설

(1) 소스 코드 1은 C 언어로 작성된 코드입니다.

(2) 소스 코드 2는 자바(Java) 언어로 작성된 코드입니다.

02-1 문자 데이터

1. ②, ④

해설

① 아라비아 숫자(0~9)로 표기한 데이터는 경우에 따라서 문자 데이터 또는 숫자 데이터로 사용될 수 있습니다.

③ 따옴표 그 자체를 문자 데이터로 사용할 수 있습니다. 역슬래시(\)를 사용해서 이스케이프 문자로 만들어 사용할 수 있습니다.

⑤ 문자 인코딩의 한 종류인 UTF-8은 전 세계적으로 사용됩니다.

2. ②

해설

② 문자 데이터는 따옴표를 양옆에 써 주어야 합니다.

3.

```
"프로그래밍을 \'처음\' 배우는 상황이라면?" 또는 '프로그래밍을 \'처음\' 배우는 상황이라면?'
"프로그래밍이란 무엇인지 \'감\'을 잡고 싶다면?" 또는 '프로그래밍이란 무엇인지 \'감\'을 잡고 싶다면?'
'프로그래밍 \'입문자\'에게 책을 추천하고 싶다면?'
'이 모든 상황에 알맞은 \"만능", \'코딩 입문서\'!\n'
"\"혼자 공부하는 첫 프로그래밍\"을 소개합니다!!!"
```

4.

```
Hello, World!
Sun is shining.
```

5.

```
"혼공프"로 시작하는 프로그래밍!
'혼공프'로 시작하는 프로그래밍!
'혼공프'로 시작하는 프로그래밍!
"혼공프"로 시작하는 프로그래밍!
Sun,
Moon,
Sky
```

해설

- 1줄: 작은따옴표('...')로 만들어진 문자 데이터는 큰따옴표("...")를 문제 없이 출력합니다(실행 결과 1줄).
- 2줄: 작은따옴표로 만들어진 문자 데이터에서 작은따옴표를 문자 데이터로 사용하려면 이스케이프 문자 \' 를 사용합니다(실행 결과 2줄).
- 3줄: 큰따옴표("...")로 만들어진 문자 데이터는 작은따옴표('...')를 문제 없이 출력합니다(실행 결과 3줄).
- 4줄: 큰따옴표로 만들어진 문자 데이터에서 큰따옴표를 문자 데이터로 사용하려면 이스케이프 문자 \" 를 사용해야 합니다(실행 결과 4줄).
- 5줄: 이스케이프 문자 \n은 문자 데이터 출력 시 새로운 줄에 출력합니다. 이스케이프 문자 \n이 두 번 사용됐으므로, 총 세 줄에 걸쳐 출력합니다(실행 결과 5~7줄).

02-2 문자 데이터 처리

1. ④

해설

① 문자 데이터의 길이를 계산할 때 하나의 공백을 1로 계산합니다.

② 문자 데이터의 길이를 계산할 때 이스케이프 문자는 역슬래시와 기호를 하나로 생각해서 1로 계산합니다.

③ 문자 데이터의 인덱스는 0부터 시작하고, 다른 번호로 시작할 수 없습니다.

⑤ 슬라이싱을 할 때 문자 데이터의 길이를 넘어선 인덱스 번호를 사용하면 자동으로 마지막 문자까지 처리합니다.

⑥ 인덱싱을 할 때는 반드시 인덱스 번호를 알맞게 적어 줘야 합니다.

2.

```
3 + 2 = 5
==========
5분 23초
```

해설

- 1줄: 5개의 문자 데이터를 문자 데이터 결합 연산자(+)로 결합한 값을 화면에 출력합니다(실행 결과 1줄).
- 2줄: 문자 데이터 "="를 문자 데이터 반복 결합 연산자(*)로 10회 반복 결합한 결괏값을 화면에 출력합니다(실행 결과 2줄).
- 3줄: 5개의 문자 데이터를 문자 데이터 결합 연산자로 결합한 결괏값을 화면에 출력합니다(실행 결과 3줄). 빈 공백(" ")도 하나의 문자로 처리됩니다.

3.

```
3

2
```

해설

- 1줄: 문자 데이터 "0123456789"를 만들고, 인덱싱 [3]을 처리한 결괏값을 화면에 출력합니다. 인덱스는 0부터 시작하므로, 인덱스 3은 네 번째 문자를 가리킵니다. 따라서 이 코드의 결괏

값은 3을 화면에 출력합니다.

- 2줄: 문자 데이터를 만들고, 슬라이싱 [3:3]을 처리한 결괏값을 화면에 출력합니다. 슬라이싱을 할 때 시작 위치 인덱스부터 (끝 위치 인덱스 – 1)까지 문자를 선택해서 자릅니다. 따라서 [3:3]과 같이 시작 위치 인덱스와 끝 위치 인덱스가 같은 슬라이싱의 경우, 결괏값으로 어떤 문자도 선택하지 않은 빈 문자 데이터(" ")를 만들게 됩니다. 그 결과 이 코드는 빈 문자 데이터를 출력합니다.
- 3줄: 문자 데이터를 만들고, 슬라이싱 [2:3]을 처리한 결괏값을 화면에 출력합니다. 슬라이싱을 할 때 끝 위치에 해당하는 인덱스는 포함하지 않기에 슬라이싱 [2:3]의 결괏값은 인덱싱 [2]와 같은 값인 "2"가 됩니다. 따라서 이 코드는 "2"를 화면에 출력합니다.

4.

```
혼자 공부하는 프로그래밍
13
혼자
공부하는
프로그래밍
```

해설

- 1줄: 문자 데이터 "혼자 공부하는 프로그래밍"을 화면에 출력합니다(실행 결과 1줄).
- 2줄: 파이썬 명령어 len을 사용해서 문자 데이터 "혼자 공부하는 프로그래밍"의 문자 개수를 화면에 출력합니다(실행 결과 2줄). 빈 공백도 하나의 문자에 포함되는 것을 기억하세요.
- 3줄: 문자 데이터 "혼자 공부하는 프로그래밍"에 슬라이싱을 적용합니다. 범위 시작 인덱스를 지정하지 않았으므로 자동으로 인덱스 0부터 시작해서 인덱스 (2–1)까지 슬라이싱하고, 그 결괏값을 화면에 출력합니다(실행 결과 3줄). 슬라이싱 할 때 범위 끝 인덱스에 해당하는 문자 데이터는 포함하지 않는다는 것을 기억하세요.
- 4줄: 문자 데이터에 슬라이싱을 적용합니다. 인덱스 3부터 인덱스 (7–1)까지 슬라이싱하고, 그 결괏값을 화면에 출력합니다(실행 결과 4줄).
- 5줄: 문자 데이터에 슬라이싱을 적용합니다. 범위 끝 인덱스를 지정하지 않았으므로, 자동으로

문자 데이터의 문자 개수 13으로 결정됩니다. 따라서 인덱스 3부터 인덱스 (13−1)까지 슬라이싱하고, 그 결괏값을 화면에 출력합니다(실행 결과 5줄).

5.

```
C Major Scale
-------------
CDEFGAB
```

해설

- 1줄: 문자 데이터 "C Major Scale"를 화면에 출력합니다(실행 결과 1줄).
- 2줄: 문자 데이터 반복 결합 연산자를 사용해서 "−"를 반복 결합합니다. 이때 결합 횟수로 문자 데이터 "C Major Scale"의 문자 개수를 지정합니다(실행 결과 2줄).
- 3−11줄: 문자 데이터 인덱싱과 문자 데이터 결합 연산자를 사용해서 "CDEFGAB"를 화면에 출력합니다(실행 결과 3줄).

02-3 숫자 데이터

1. ④

해설

① "323"과 323은 아라비아 숫자가 쓰였지만, "323"은 숫자 기호가 따옴표로 쌓여있기 때문에 문자 데이터로 처리됩니다.

② 코딩에서 여러 가지 연산이 연속으로 사용된 경우 소괄호(...)를 사용해서 연산의 우선순위를 강제로 정할 수 있습니다. 코딩에서 대괄호[...]와 중괄호{...}는 다른 의미로 사용합니다.

③ 여러 개의 사칙 연산자(+, −, *, /)가 동시에 사용된 경우 곱셈과 나눗셈이 덧셈과 뺄셈보다 우선 처리됩니다.

⑤ 나머지 연산자(%)는 숫자 데이터의 나눗셈에서 '몫과 나머지' 방식으로 계산한 뒤, 나머지를 선택하는 기능을 합니다. 따라서 (3%2)의 결괏값은 1이 됩니다. 책의 본문을 참고하세요.

2.

```
숫자 데이터(정수) 확인 문제
=*=*=*=*=*=
5
1
6
1.5
9
5
23
```

해설

- 1줄: "숫자 데이터(정수) 확인 문제"를 화면에 출력합니다(실행 결과 1줄).
- 2줄: "=*"를 반복 결합 연산자(*)를 사용해서 5회 반복 결합하고, 결합 연산자(+)를 사용해서 "="를 결합한 뒤, 그 결괏값을 화면에 출력합니다(실행 결과 2줄).
- 3~6줄: 숫자 데이터 3과 2에 대해서 사칙 연산(덧셈, 뺄셈, 곱셈, 나눗셈)한 결괏값을 화면에 출력합니다(실행 결과 3~6줄).
- 7줄: 숫자 데이터 3을 2 제곱한 결괏값을 화면에 출력합니다(실행 결과 7줄).
- 8줄: 숫자 데이터 323을 60으로 나눈 몫을 화면에 출력합니다(실행 결과 8줄).
- 9줄: 숫자 데이터 323을 60으로 나눈 나머지를 화면에 출력합니다(실행 결과 9줄).

3.

```
숫자 데이터(부동소수점수) 확인 문제
=*=*=*=*=*=
383.0
263.0
19380.0
5.383333333333334
3.5657536863853114e+150
5.0
23.0
```

해설

- 1~2줄: 1번 문제와 동일합니다(실행 결과 1~2줄).
- 3~6줄 : 부동소수점수 323.0과 정수 60에 대해서 사칙 연산한 결괏값을 화면에 출력합니다(실행 결과 3~6줄).
- 7줄: 부동소수점수 323.0을 60 제곱한 결괏값을 화면에 출력합니다. 파이썬은 컴퓨터 메모리 용량 한도 내에서 무한대의 수를 표현할 수 있습니다(실행 결과 7줄).
- 8줄: 부동소수점수 323.0을 60으로 나눈 몫을 화면에 출력합니다(실행 결과 8줄).
- 9줄: 부동소수점수 323.0을 60으로 나눈 나머지를 화면에 출력합니다(실행 결과 9줄).

4.

```
연산자 우선순위 확인 문제
=*=*=*=*=*=
523
523
500
```

해설

- 1~2줄: 1번 문제와 동일합니다(실행 결과 1~2줄).
- 3줄: 곱셈과 나눗셈 연산자는 덧셈과 뺄셈 연산자에 대해서 우선순위가 있습니다. 따라서 수식 323 // 60 * 100이 먼저 처리되고, 그 결괏값에 23을 더한 값을 화면에 출력합니다(실행 결과 3줄).
- 4줄: 괄호는 다른 어떤 연산자보다 먼저 처리됩니다. 따라서 수식 (323 // 60 * 100)이 먼저 처리됩니다(실행 결과 4줄).
- 5줄: 괄호가 먼저 처리되므로 (23 + 323)이 처리된 후, 그 결괏값에 // 60 * 100이 처리됩니다(실행 결과 5줄).

02-4 변수

1. "안녕하세요"는 변수 이름으로 사용할 수 있지만(네이밍 룰 만족), 특별한 이유가 없다면 사용하지 않는 것이 좋습니다(네이밍 컨벤션 불만족).

변수 이름	사용 가능 여부(O/X)	사용 불가능한 이유
this_element	O	-
that element	X	공백을 포함할 수 없습니다.
$my_element	X	특수문자는 밑줄(_)만 사용할 수 있습니다.
_	O	-
None	X	키워드는 사용할 수 없습니다.
안녕하세요	O	-

2.

```
Hello, World!
```

해설

- 1줄: 변수 hello 에 문자 데이터 "Hello"를 저장합니다.
- 2줄: 변수 world 에 문자 데이터 "World!"를 저장합니다.
- 3줄: 문자 데이터 결합 연산자를 사용해서 hello, 문자 데이터 ", ", world를 결합한 결괏값을 화면에 출력합니다(실행 결과 1줄).

3.

```
5
23
```

해설

- 1줄: 숫자 데이터 323을 60으로 나눈 몫을 변수 quotient에 저장합니다.
- 2줄: 숫자 데이터 323을 60으로 나눈 나머지를 변수 remainder에 저장합니다.
- 3~4줄: quotient와 remainder에 저장된 데이터를 각각 화면에 출력합니다(실행 결과 1!2줄).

4.

```
안녕하세요
나한빛입니다
```

해설

- 1줄: "안녕하세요" 문자 데이터를 만들고 변수 data에 저장합니다.
- 2줄: print 명령어를 사용해서 변수 data에 저장된 데이터, 즉 "안녕하세요"를 화면에 출력합니다.
- 3줄: "만나서 반가워요" 문자 데이터를 만들고 변수 data에 저장합니다. 변수는 데이터를 반복해서 저장할 수 있습니다.
- 4줄: "나한빛입니다" 문자 데이터를 만들고 변수 data에 저장합니다.
- 5줄: print 명령어를 사용해서 변수 data에 저장된 데이터를 화면에 출력합니다. 현재 data에 저장된 데이터는 "나한빛입니다"이므로 이 데이터가 화면에 출력됩니다.

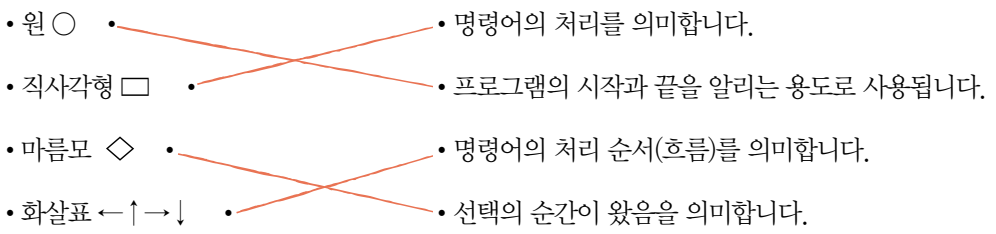
03-1 선택 구조 이해하기

1. ④

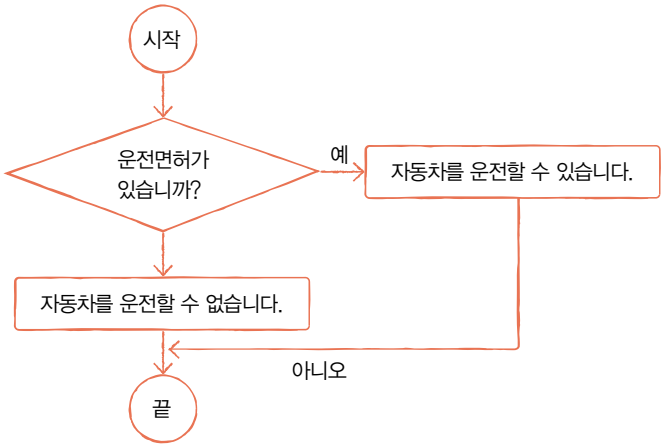
해설

④ 조건식을 그림을 표현할 때 조건식의 결과가 "예"인 경우 반드시 오른쪽 화살표로, "아니오"인 경우 반드시 아래쪽 화살표로 그려야 하는 것은 아닙니다. 프로그램의 진행 방향에 따라 다양한 방향의 화살표를 사용할 수 있습니다.

2.



3.



03-2 선택 구조를 파이썬으로 코딩하는 방법

1.

```
True
False
True
True
```

해설

- 1줄: 불 데이터 True와 True는 같은 값이므로, 이 조건식의 결괏값은 True 입니다(실행 결과 1줄).
- 2줄: 문자 데이터 "True"와 불 데이터 True는 다른 값이므로, 이 조건식의 결괏값은 False입니다(실행 결과 2줄).
- 3줄: 숫자 데이터 323을 60으로 나눈 몫은 5이므로, 이 조건식의 결괏값은 True입니다(실행 결과 3줄).
- 4줄: 문자 데이터끼리의 크기 비교는 이름순 정렬과 같습니다. 프로그래밍에서 영어 소문자는 대문자보다 이름순 정렬 시 우선순위를 가지므로, 문자 데이터 "abc"는 문자 데이터 "Abc"보다 큽니다. 따라서 이 조건식의 결괏값은 True입니다(실행 결과 4줄).

2.

```
시작!
예!
끝!
```

해설

- 1줄: "시작!" 메시지를 화면에 출력합니다(실행 결과 1줄).
- 2~5줄: if~else 구조를 시작합니다. 조건식의 결괏값이 True인 경우 3줄을 처리하고, 그렇지 않으면(else) 5줄을 처리합니다. 2줄에서는 조건식에 직접 True 값이 주어졌으므로 3줄을 처리합니다(실행 결과 2줄).
- 6줄: "끝!" 메시지를 화면에 출력합니다(실행 결과 3줄).

3.

```
시작!
취미로 운동을 하는군요!
끝!
```

해설

- 1줄: 문자 데이터 "운동"을 변수 hobby에 저장합니다.
- 3줄: 문자 데이터 "시작!"을 화면에 출력합니다(실행 결과 1줄).
- 4~11줄: if~elif~else 구조를 시작합니다. 모두 4개의 대안이 있고, 3개의 조건식이 있는 구조입니다.
- 4줄: 변수 hobby에 저장된 값이 문자 데이터 "독서"와 같다면 5줄의 명령문을 실행하고, if~elif~else 구조를 종료합니다. 1줄에서 hobby에 저장된 값이 "운동"이므로, 4줄의 조건식의 결괏값은 False입니다. 따라서 5줄의 무시하고 6줄로 넘어갑니다.
- 6줄: hobby에 저장된 데이터가 "운동"이면, 7줄의 명령어를 실행합니다. 1줄에서 hobby에 저장된 데이터가 "운동"이므로, 6줄의 조건식의 결괏값은 True입니다. 따라서 7줄을 실행하고 if~elif~else 구조를 종료합니다.
- 7줄: 문자 데이터 "취미로 운동을 하는군요!"를 화면에 출력합니다(실행 결과 2줄).

• 12줄: 문자 데이터 "끝!"을 화면에 출력해서 프로그램의 종료를 알립니다.

04-1 효율적인 데이터 관리 1: 리스트

1. ①

해설

② 여러 개의 데이터를 하나의 데이터 세트로 만드는 방법은 다양합니다. 이 책에서는 리스트, 딕셔너리, 레인지 데이터 세트를 소개합니다.

③ 프로그래밍 언어에 따라 리스트를 만들 때 다양한 괄호를 사용합니다. 예를 들어 파이썬과 자바스크립트는 대괄호[...]를, C 언어와 자바는 중괄호{...}를 사용합니다. ④ 리스트를 만들 때 각 데이터 간의 구분은 쉼표(,)를 사용합니다.

⑤ 리스트에 저장된 각 데이터와 쉼표 사이에 공백을 자유롭게 추가할 수 있습니다. 일반적으로 시각적인 편의를 위해서 데이터 뒤에 붙는 쉼표는 공백이 없고, 데이터 앞의 쉼표는 1칸의 공백을 추가합니다.

2.

```
lang_set     = ["파이썬", "C 언어", "자바", "자바스크립트"]
lang_eng_set = ["Python", "C", "Java", "JavaScript"]
```

3.

```
menu_set  = ["안심구이", "등심구이", "육개장", "물냉면"]
price_set = [39000, 35000, 9000, 8000]
```

04-2 효율적인 데이터 관리 2: 딕셔너리

1. 리스트를 사용한 성적표 데이터 관리

```
grade    = ["1학년", "2학년", "3학년"]
subjects = ["국어", "영어", "수학", "코딩", "체육"]
scores   = [
```

```
    [85, 0, 60, 95, 0],
    [90, 80, 70, 95, 0],
    [95, 85, 75, 100, 100],
]
```

해설

이 코드는 리스트를 사용해서 성적표 데이터를 정리한 것입니다. 리스트로 데이터를 관리할 때 주의할 점은, 데이터 세트 별로 동일한 종류 및 동일한 개수의 데이터를 사용해야 한다는 것입니다.

변수 scores에 저장된 2차원 리스트를 살펴보죠. 이 문제에서 영어 과목은 2학년에 처음 등장했지만, 데이터 세트별로 데이터 개수를 맞추기 위해 1학년 데이터에도 영어 점수를 추가(0점)했습니다. 마찬가지로 3학년에 처음 등장한 체육 과목을 위해서 1학년 및 2학년에 체육 과목 점수를 추가(0점)해야 합니다.

만약, 학년이 올라갈수록 과목이 증가하면 어떤 코드를 어떻게 바꿔야 할까요? 한번 고민해 보세요.

설명을 간단히 하기 위해서 영어 점수를 1학년에 추가할 때 0점으로 넣었지만, 이런 경우 왜곡된 정보(실제 0점인지, 과목이 없었는지 구분하기 어려움)를 만들 수 있습니다. 따라서 0점을 넣는 것보다 '데이터 없음'이라는 표시를 하는 게 좋습니다. 프로그래밍 언어별로 '데이터 없음'을 표시하는 방법이 다양한데요, 파이썬은 None이라는 키워드를 사용합니다.

딕셔너리를 사용한 성적표 데이터 관리 <1>

```
scores = {
    "1학년": [85, 0, 60, 95, 0],
    "2학년": [90, 80, 70, 95, 0],
    "3학년": [95, 85, 75, 100, 100],
}
```

해설

이 코드는 앞서 리스트로 만든 코드에 학년 정보만 이름표로 추가해서 딕셔너리로 변환한 것입니

다. 학년별 점수 데이터 세트는 리스트를 계속 사용하기 때문에 리스트의 단점, 즉 불필요한 정보(1학년 영어 및 체육 과목 점수, 2학년 체육 과목 점수)를 관리해줘야 문제가 계속 발생하죠.

딕셔너리를 사용한 성적표 데이터 관리 〈2〉

```
scores = {
    "1학년": { "국어": 85, "수학": 60, "코딩": 95 },
    "2학년": { "국어": 90, "영어": 80, "수학": 70, "코딩": 95 },
    "3학년": { "국어": 95, "영어": 85, "수학": 75, "코딩": 100, "체육": 100 }
}
```

해설

이 코드는 학년별 점수 데이터 세트도 딕셔너리로 표현했습니다. 이렇게 함으로써 리스트로 관리했을 때의 단점을 해결할 수 있습니다.

딕셔너리를 사용한 성적표 데이터 관리 〈3〉

```
scores = {
    "국어": { "1학년": 85, "2학년": 90, "3학년": 95 },
    "수학": { "1학년": 60, "2학년": 70, "3학년": 75 },
    "코딩": { "1학년": 95, "2학년": 95, "3학년": 100 },
    "영어": { "2학년": 80, "3학년": 85 },
    "체육": { "3학년": 100 }
}
```

해설

이 코드는 딕셔너리를 사용해서 과목별, 학년별로 데이터 세트를 구성했습니다. 위 〈2〉번 코드와 비교했을 때, 어떤 기준을 우선순위로 둘 것인지에 대한 관점(학년별, 과목별 vs 과목별, 학년별)이 달라졌습니다. 어느 것이 더 좋다고 말할 수 없고, 필요에 따라 적절하게 선택하면 됩니다.

2. 리스트를 사용한 주식 가격 데이터 관리 〈1〉

```
names = ["삼송전자", "헤이닉스"]
prices = [
    [50800, 49950, 48900, 47300, 45600, 48100],
    [82800, 82500, 80600, 80400, 73100, 83100]
]
```

해설

이 코드에서 가격 데이터 세트를 관리하는 방법을 잘 살펴보세요. 여기서는 '종목별로, 일자별로' 가격 데이터를 관리합니다.

리스트를 사용한 주식 가격 데이터 관리 〈2〉

```
names = ["삼송전자", "헤이닉스"]
prices = [
    [50800, 82800],
    [49950, 82500],
    [48900, 80600],
    [47300, 80400],
    [45600, 73100],
    [48100, 83100]
]
```

해설

이 코드는 가격 데이터 세트를 '일자별로, 종목별로' 관리한 것입니다.

앞서 '리스트를 사용한 주식 가격 데이터 관리 〈1〉'과 비교했을 때 어느 것이 좋다고 말하기 어렵습니다. 어떤 프로그램을 작성할 것인가에 따라서 적절히 선택해서 사용하면 됩니다.

정답 및 해설

딕셔너리를 사용한 주식 가격 데이터 관리 〈1〉

```
prices = {
    "삼송전자": [50800, 49950, 48900, 47300, 45600, 48100],
    "헤이닉스": [82800, 82500, 80600, 80400, 73100, 83100]
}
```

해설

이 코드는 딕셔너리를 사용해서 종목별로, 일자별로 가격 데이터를 관리한 것입니다. 앞서 '리스트를 사용한 주식 가격 데이터 관리 〈1〉' 코드에서 names와 prices를 하나로 합친 것이라고 볼 수 있죠. 일자별 가격 데이터 세트는 리스트를 사용했습니다.

딕셔너리를 사용한 주식 가격 데이터 관리 〈2〉

```
prices = {
    "삼송전자": {
        "2": 50800,
        "3": 49950,
        "4": 48900,
        "5": 47300,
        "6": 45600,
        "9": 48100
    },
    "헤이닉스": {
        "2": 82800,
        "3": 82500,
        "4": 80600,
        "5": 80400,
        "6": 73100,
        "9": 83100
    }
}
```

해설

이 코드는 일자별 가격 데이터 세트를 딕셔너리로 관리한 것입니다. '딕셔너리를 사용한 주식 가격 데이터 관리 〈1〉'의 코드와 비교했을 때, 날짜별 가격을 확인할 수 있다는 것이 특징이죠.

3. 리스트를 사용한 요리별 재료 데이터 관리

```
names = ["김치찌개", "떡볶이"]
recipe = [
    ["돼지고기", "김치", "마늘", "마늘", "청양고추"],
    ["떡", "설탕", "고추장", "간장", "고춧가루", "대파", "어묵"],
]
```

딕셔너리를 사용한 요리별 재료 데이터 관리

```
recipe = {
    "김치찌개": ["돼지고기", "김치", "마늘", "마늘", "청양고추"],
    "떡볶이": ["떡", "설탕", "고추장", "간장", "고춧가루", "대파", "어묵"],
}
```

4. 딕셔너리를 사용한 이력서 데이터 관리

```
items = {
    "성명(한글)": "나한빛",
    "성명(영문)": "Na Hanbit",
    "주소": "서울시 서대문구",
    "취미 및 특기": {"취미": "독서", "특기": "글쓰기"},
    "저서": {
        "혼공 시리즈": ["혼공프로", "혼공파", "혼공씨", "혼공자바"],
        "이것이 시리즈": ["이것이 데이터 분석이다 with 파이썬", "이것이 C 언어다", "이
것이 자바다"],
    },
}
```

05-1 반복 알아보기

1.

```
5!
4!
3!
2!
1!
0!
```

해설

- 1줄: count 변수에 5를 저장합니다. 이 변수는 앞으로 반복 처리 횟수를 관리하는 도구로 사용됩니다.
- 2줄: while 반복문을 사용해서 조건식 'count 값이 0보다 크거나 같은가요?'의 결괏값이 True인 경우, 3~4줄을 반복 처리합니다. 현재 count는 5를 저장하고 있으므로, 이 조건식의 결괏값은 True가 되고, 3~4줄을 처리할 수 있습니다.
- 3줄: count 변수에 저장된 숫자를 문자 데이터로 변환하고(str 명령어 사용), "!" 문자 데이터와 연결한 결괏값을 화면에 출력합니다.
- 4줄: count 변수에서 1을 뺀 값을 count 변수에 다시 저장합니다. 이제 count는 4를 저장하고, 2줄로 돌아가서 조건식의 결괏값에 따라 3~4줄을 반복 처리할지 결정합니다. 이런 식으로 반복 처리를 할 때 중요한 것은 조건식에서 사용한 비교 연산자의 올바른 선택입니다.
- 이 예제에서는 '>='를 사용했기 때문에 count 값이 5부터 0이 될 때까지 총 6회 반복 처리하지만, 만약 '>'를 사용하면 count 값이 5부터 1이 될 때까지 총 5번 반복 처리합니다. 이처럼 조건식이 조금만 달라져도 프로그램의 결과는 크게 차이가 나기 때문에, while 반복문을 사용할 때는 항상 조건식의 정확한 관리가 중요합니다.

2.

```
5!
4!
3!
2!
1!
```

해설

- 1줄: 5부터 1까지 숫자 데이터를 저장한 리스트 데이터 세트를 만들고 변수 count에 저장합니다.
- 2줄: for 반복문을 사용해서 count에 저장된 데이터 개수만큼 3줄을 반복 처리합니다. 변수 x에는 매회 반복 처리할 때마다 count 데이터 세트에서 꺼낸 데이터가 저장됩니다.
- 3줄: x에 저장된 숫자 데이터를 문자 데이터로 변환하고, "!" 문자 데이터를 연결한 결괏값을 화면에 출력합니다.
- 이처럼 여러분이 의도한 대로 특정 횟수만큼 정확히 반복 처리하기 위해서는, 반복 처리할 횟수만큼의 데이터를 저장한 데이터 세트를 미리 준비하고, for 반복문을 사용해서 처리하는 것이 안전합니다.

3.

```
1
2
짝!
4
5
짝!
7
8
짝!
10
```

해설

- 1줄: range 명령어를 사용해서 10개의 숫자 데이터를 가진 레인지 데이터 세트를 만들고, count 변수에 저장합니다.
- 2줄: for 반복문을 사용해서 count 데이터 세트에 저장된 데이터 개수만큼 3~6줄을 반복 처리합니다. 변수 n은 매회 반복 처리할 때마다 0부터 9까지 숫자가 저장됩니다.
- 3줄: if~else 조건문을 사용해서 조건식 '(n + 1) % 3 == 0'의 결괏값이 True라면 4줄을 실행하고, False라면 6줄을 실행합니다. 이 조건식은 '현재 실행 횟수가 3의 배수인가요?'라는 의미입니다. 단, n이 0부터 9까지 저장되기 때문에, 1을 더해서 1부터 10까지 저장된 변수처럼 활용합니다.
- 4줄: 화면에 "짝!" 메시지를 출력합니다. 3줄의 조건식의 결괏값이 True일 때 실행됩니다.
- 5줄: 3줄의 조건식의 결괏값이 False일 때 실행됩니다. else는 별다른 처리를 하지 않고, 바로 다음 줄의 코드를 실행합니다.
- 6줄: 변수 n에 1을 더한 결괏값을 화면에 출력합니다.
- 이 프로그램은 369 게임을 for 반복문을 사용해서 만든 것입니다. 첫 번째 반복 시에는 n에 0이 저장되고, 따라서 조건식의 결과는 False 가 되므로 6줄을 실행합니다. 총 10회 반복 할 때마다 n 변수, 조건식 결괏값, 처리되는 명령어, 화면 출력값을 정리하면 다음 표와 같습니다.

반복 횟수	n	(n + 1) % 3 != 0	처리되는 명령어	화면 출력값
1	0	True	print(n + 1)	1
2	1	True	print(n + 1)	2
3	2	False	print("짝!")	짝!
4	3	True	print(n + 1)	4
5	4	True	print(n + 1)	5
6	5	False	print("짝!")	짝!
7	6	True	print(n + 1)	7
8	7	True	print(n + 1)	8
9	8	False	print("짝!")	짝!
10	9	True	print(n + 1)	10

4.

```
혼자
공부하는
첫 프로그래밍!
```

해설

- 1줄: 5개의 문자 데이터를 저장한 리스트 데이터 세트를 만들고, words 변수에 저장합니다.
- 2줄: for 반복문을 사용해서 words 데이터 세트에 저장된 데이터 개수만큼 3~6줄을 반복 처리합니다. 변수 x는 매회 반복 처리할 때마다 "혼자", "공부하는", "첫", "프로그래밍", "!" 문자 데이터가 저장됩니다.
- 3줄: if 조건문을 사용해서 조건식 'x == "첫"'의 결괏값이 True일 때 4~5줄을 처리하고, False일 때는 6줄을 처리합니다. 이 조건식은 'x에 저장된 데이터가 "첫" 문자 데이터인가요?'라는 의미입니다.
- 4줄: 만약 x에 저장된 데이터가 "첫" 문자 데이터라면, "첫 프로그래밍!"을 화면에 출력하고, 5줄을 실행합니다.
- 5줄: break 명령어를 사용해서 현재 진행 중인 반복 처리를 중단하고, 전체 for 반복문을 종료합니다. 따라서 5줄의 명령어가 실행되면 6줄은 처리되지 않습니다.
- 6줄: x에 저장된 데이터가 "첫" 문자 데이터가 아니라면, x에 저장된 데이터를 화면에 출력합니다. 따라서 "혼자", "공부하는" 등 2개의 문자 데이터만 화면에 출력하고, x에 "첫"이 저장되면 3줄의 조건식이 True가 되므로, 4~5줄을 실행 후 for 반복문을 종료합니다.

5.

```
10!
20!
```

해설

- 1줄: range 명령어를 사용해서 20개의 숫자 데이터를 저장한 레인지 데이터 세트를 만들고, count 변수에 저장합니다.

- 2줄: for 반복문을 사용해서 count 데이터 세트에 저장된 데이터 개수만큼 3~5줄을 반복 처리합니다. 변수 x는 매회 반복 처리할 때마다 0부터 19까지 20개의 데이터가 차례대로 저장됩니다.
- 3줄: if 조건문을 사용해서 조건식 '((x + 1) % 10) != 0'의 결괏값이 True인 경우 4줄을 처리하고, False인 경우 5줄을 처리합니다. 이 조건식의 의미는 '현재 진행 중인 반복 횟수가 10의 배수가 아닌가요?' 변수 x는 0부터 시작하기 때문에 1부터 시작하는 횟수로 바꾸기 위해서 x를 (x + 1)로 바꿨습니다.
- 4줄: continue 명령어를 사용해서 현재 진행 중인 반복 처리를 중단하고, 다음 횟수의 반복 처리를 합니다.
- 5줄: 현재 진행 중인 반복 처리 횟수를 문자 데이터로 변환하고, "!" 문자 데이터와 연결한 결괏값을 화면에 출력합니다.

05-2 데이터 세트와 for 반복문

1.

```
플
랫
화
이
트
```

해설

- 1줄: "플랫화이트" 문자열(여러 개의 문자를 하나의 세트로 만든 문자 데이터)을 만들고 coffee 변수에 저장합니다.
- 2줄: for 반복문을 사용해서 coffee 문자열의 문자 개수만큼 3줄을 반복 처리합니다. 변수 x에는 매회 반복 처리할 때마다 "플", "랫", "화", "이", "트"가 저장됩니다.
- 3줄: 변수 x에 저장된 문자 데이터를 화면에 출력합니다.

2.

```
2!
4!
6!
8!
10!
```

해설

- 1줄: 1부터 10까지 숫자 데이터를 저장한 리스트를 만들고 count 변수에 저장합니다.
- 2줄: for 반복문을 사용해서 count 데이터 세트에 저장된 데이터 개수만큼 3~4줄을 반복 처리합니다. 변수 x에는 매회 반복 처리할 때마다 1부터 10까지 숫자 데이터가 저장됩니다.
- 3줄: if 조건문으로 조건식 'x % 2 == 0'의 결괏값이 True일 경우 4줄을 실행합니다. 이 조건식의 의미는 'x가 2의 배수인가요?'입니다. 따라서 x가 2의 배수일 때만 4줄을 실행합니다.
- 4줄: x에 저장된 숫자 데이터를 문자 데이터로 바꾸고, "!" 문자를 연결한 결괏값을 화면에 출력합니다.

3.

```
3
6
9
12
15
```

해설

- 1줄: range 명령어를 사용해서 0부터 1씩 증가하는 숫자 데이터를 5개 저장한 레인지 데이터 세트를 만들고 five 변수에 저장합니다.
- 2줄: for 반복문을 사용해서 five 데이터 세트의 데이터 개수만큼 3줄을 반복 처리합니다. 이때 변수 x에는 매회 반복 처리할 때마다 0, 1, 2, 3, 4가 차례대로 저장됩니다.
- 3줄: 변수 x에 저장된 값에 1을 더하고, 3을 곱한 결괏값을 화면에 출력합니다. 이 코드는 five에 저장된 데이터 개수만큼 반복 처리되는데, 결국 3의 배수를 5번 출력하게 됩니다.

4.

```
아메리카노 : 3100
플랫 화이트 : 4100
화이트 초콜릿 모카 : 4600
```

해설

- 1줄: 커피 이름을 담은 문자 데이터 3개를 리스트로 만들고, order 변수에 저장합니다.
- 2줄: 커피 가격을 담은 숫자 데이터 3개를 리스트로 만들고, price 변수에 저장합니다.
- 3줄: for 반복문을 사용해서 데이터 개수(3개)만큼 4줄을 반복 처리합니다. 이때 for 반복문의 반복 횟수를 결정할 데이터 세트로 range(3) 코드로 만든 레인지 데이터 세트를 사용합니다. 변수 x는 매회 반복 처리할 때마다 0, 1, 2가 저장됩니다.
- 4줄: x에 저장된 인덱스 번호를 사용해서 order 리스트와 price 리스트에 저장된 데이터를 하나씩 꺼내 화면에 출력합니다.

5.

```
990
120
990
120
```

해설

- 1줄: 숫자 데이터 2개를 저장한 리스트를 만들고, scores 변수에 저장합니다.
- 2줄: scores 데이터 세트에서 0번 인덱스에 저장된 데이터를 화면에 출력합니다. 리스트는 인덱스로 데이터에 접근할 수 있다는 사실을 기억하세요.
- 3줄: scores 데이터 세트에서 1번 인덱스에 저장된 데이터를 화면에 출력합니다.
- 5줄: 숫자 데이터 2개에 각각 "TOEIC", "TOEFL iBT" 이름표를 붙여서 딕셔너리를 만들고, scores 변수에 저장합니다.
- 6줄: scores 데이터 세트에서 "TOEIC" 이름표가 붙은 데이터를 화면에 출력합니다. 딕셔너리는 이름표로 데이터에 접근할 수 있다는 사실을 기억하세요.

• 7줄: scores 데이터 세트에서 "TOEFL iBT" 이름표가 붙은 데이터를 화면에 출력합니다.

6.

```
아메리카노 : 3100
플랫 화이트 : 4100
화이트 초콜릿 모카 : 4600
```

해설

• 1줄: 커피 이름을 담은 문자 데이터 3개를 리스트로 만들고, order 변수에 저장합니다.

• 2~6줄: 커피 가격을 담은 숫자 데이터 3개에 각각 이름표를 붙인 딕셔너리를 만들고, price 변수에 저장합니다.

• 8줄: for 반복문을 사용해서 order 데이터 세트의 데이터 개수만큼 9줄을 반복 처리합니다. 변수 x는 매회 반복 처리할 때마다 "아메리카노", "플랫 화이트", "화이트 초콜릿 모카"가 저장됩니다.

• 9줄: x에 저장된 문자 데이터를 이름표로 사용해서 price 딕셔너리에서 가격 데이터를 꺼내고, 이름표와 함께 화면에 출력합니다.

7.

```
6
15
```

해설

• 1줄: 리스트의 데이터로 다른 리스트를 저장한 2차원 리스트를 만들고 변수 numbers에 저장합니다. 2차원 리스트는 이 책의 [04-2]절에서 다루었습니다. numbers 리스트에 저장된 첫 번째 데이터는 [1, 2, 3] 리스트가 되고, 두 번째 데이터는 [4, 5, 6] 리스트가 됩니다.

• 3줄: for 반복문을 사용해서 numbers 리스트에 저장된 데이터 개수만큼 반복 처리합니다. 이때 매회 반복 시마다 변수 row에는 [1, 2, 3]과 [4, 5, 6] 리스트가 차례대로 저장됩니다.

• 4줄: 숫자 데이터 0을 만들고 변수 total에 저장합니다. total 변수는 3줄의 for 반복문 내부(코드 블록)에서 만들었기 때문에 4~7줄에서 사용 가능합니다. 이 변수는 앞으로 row에 저장된 리스트 데이터의 합계를 저장할 것입니다.

- 5줄: 새로운 for 반복문을 사용해서 row 리스트에 저장된 데이터 개수만큼 반복 처리합니다. 이때 매회 반복 시마다 변수 x에는 1, 2, 3(row 리스트 값이 [1, 2, 3]일 때) 또는 4, 5, 6(row 리스트 값이 [4, 5, 6]일 때)이 차례대로 저장됩니다.
- 6줄: 4줄에서 만든 total 변수에 5줄의 x에 저장된 숫자 데이터를 저장합니다. 3줄의 row 리스트 값이 [1, 2, 3]일 때 x는 1, 2, 3이 되므로 total에는 총 6이 저장되고, row 리스트 값이 [4, 5, 6]일 때 x는 4, 5, 6이 되므로 total에는 총 15가 저장됩니다.
- 7줄: total 변수에 저장된 값을 화면에 출력합니다.

8.

```
플랫 화이트 2잔, 합계 : 8200
화이트 초콜릿 모카 1잔, 합계 : 4600
```

해설

- 1~7줄: 커피 메뉴별 이름과 가격을 딕셔너리로 만들고, menu 변수에 저장합니다.
- 9~12줄: 커피 이름과 주문량을 딕셔너리로 만들고, my_order 변수에 저장합니다.
- 14줄은 for 반복문을 사용해서 my_order 데이터 세트에 저장된 데이터 개수(2개)만큼 15~18줄을 반복 처리합니다. 변수 x는 매회 반복 처리할 때마다 "플랫 화이트", "화이트 초콜릿 모카"를 저장합니다. 딕셔너리를 for 반복문에 사용하는 경우, 딕셔너리 데이터의 이름표가 x 변수에 저장된다는 사실을 기억해 주세요(리스트는 데이터 그 자체가 저장됨).
- 15줄: x에 저장된 메뉴 이름을 활용하여 menu에서 가격을 뽑아서 price 변수에 저장합니다.
- 16줄: x에 저장된 메뉴 이름을 활용하여 my_order에서 주문량을 뽑아서 qty 변수에 저장합니다.
- 17줄: 가격(price)과 주문량(qty)을 곱셈한 결괏값을 total 변수에 저장합니다.
- 18줄: 메뉴 이름(x), 주문량(qty), 총 주문가격(total)을 화면에 출력합니다.

06-1 함수 활용하기

1.

```
(1) 함수 헤더 : def add(first, second)
(2) 함수 보디 : return first + second
(3) 매개변수 : first, second
(4) 인수 : 3, 2
(5) 리턴값 : 5
```

2.

```
01:  None
02:  None
```

해설

- 1~2줄: 함수 no_return은 return 명령어를 사용하지 않는 함수입니다. 따라서 리턴값이 없는 함수입니다. 리턴값이 없는 함수는 자동으로 None을 리턴합니다.
- 4~5줄: 함수 no_return_value는 return 명령어만 사용한 함수입니다. 이러한 경우 파이썬은 자동으로 None을 리턴합니다.
- 7~8줄: no_return 함수와 no_return_value 함수는 모두 None을 리턴하므로, 각 리턴값인 None을 화면에 출력합니다(실행 결과 1, 2줄).

3.

```
Sun
is
rising
```

해설

- 1~3줄: 함수 print_element는 함수 호출 시 전달받은 인수를 매개변수 arg에 저장하고, for 구조를 활용해서 매개변수 arg에 저장된 요소를 하나씩 꺼내 화면에 출력하는 처리를 반복합니다.

- 5줄: 함수 print_element를 호출하면서 인수로 "Sun", "is", "rising" 문자 데이터를 요소로 갖는 리스트를 전달합니다. 이 인수는 함수 print_element의 매개변수 arg에 전달되고, 함수 보디 2~3줄에 의해 각 요소가 화면에 출력됩니다(실행 결과 1~3줄).

4.

```
9
0.01
```

해설

- 1~2줄: 함수 power를 정의합니다. 이 함수는 매개변수 base에 전달된 값을 매개변수 exp만큼 제곱한 값을 리턴합니다. 즉 base_exp 값을 구하는 함수죠.
- 4줄: 함수 power를 호출하면서 인수로 3과 2를 전달합니다. 따라서 3의 2제곱인 9를 리턴값으로 얻을 수 있고, 이 값을 화면에 출력합니다(실행 결과 1줄).
- 5줄: 함수 power를 호출하면서 인수로 10과 −2를 전달합니다. 따라서 10의 −2제곱인 0.01을 리턴값으로 얻을 수 있고, 이 값을 화면에 출력합니다(실행 결과 2줄).

5.

```
[9, 4, 25]
[104329, 3600]
```

해설

- 1~5줄: 함수 square는 주어진 인수의 각 요소를 제곱한 값을 리턴하는 함수입니다.
- 2줄: 결괏값을 저장할 목적으로 빈 리스트 데이터 세트를 만들고, 변수 result에 저장합니다.
- 3줄: 매개변수 dataset에는 함수 호출 시 전달된 인수가 저장됩니다. for 구조를 사용해서 dataset에 저장된 요소를 하나씩 변수 x에 저장하고 4줄의 명령어를 반복 처리합니다.
- 4줄: x를 2번 곱해서 제곱한 값(x * x)을 result 데이터 세트에 추가합니다. 리스트 데이터 세트 명령어 append는 인수로 전달된 데이터를 리스트의 끝에 추가하는 기능을 합니다. 4줄의 명령어는 arg에 저장된 요소 개수만큼 반복 처리됩니다.
- 5줄: result에 저장된 리스트 데이터 세트를 리턴합니다.

• 7줄: 함수 square를 호출하면서 [3, 2, 5]를 인수로 전달합니다. sqaure는 리스트 데이터 세트의 각 요소를 제곱한 값을 리턴하므로, [9, 4, 25]를 화면에 출력합니다(실행 결과 1줄).

• 8줄: 함수 호출 시 [323, 60]을 인수로 전달하고, 그 결과 [104329, 3600]을 화면에 출력합니다(실행 결과 2줄).

6.

```
[['my_key', 'my_value'], ['your_key', 'your_value']]
```

해설

• 1줄 : 함수 dict_to_list는 인수로 전달된 딕셔너리 데이터 세트를 리스트 데이터 세트로 변환하는 기능을 하는 함수입니다.

• 2줄 : 이 함수의 결괏값을 저장할 리스트 데이터 세트를 만들고, 변수 result 에 저장합니다.

• 3줄 : 함수 호출 시 인수로 전달된 데이터는 매개변수 dataset에 저장됩니다. for 구조를 활용해서 dataset 의 각 요소를 하나씩 꺼내 반복 처리합니다.

• 4줄 : 변수 x는 딕셔너리 데이터 세트 dataset에 저장된 각 요소의 키(key)를 의미하고, 명령어 dataset[x] 은 각 요소의 값(value)을 의미합니다. 딕셔너리 데이터 세트의 각 요소를 [키, 값]의 형식으로 리스트 데이터 세트로 변환하고([x, dataset[x]]), 리스트 데이터 세트 명령어 append를 사용해서 이 데이터를 result에 추가합니다.

• 7줄 : 함수 호출 시 인수로 사용할 딕셔너리 데이터 세트를 만들고, 변수 data에 저장합니다. 이 딕셔너리 데이터 세트는 2개의 요소를 가지고 있고, 각 요소의 키(key)는 "my_key", "your_key" 문자 데이터입니다.

• 8줄 : dict_to_list 함수를 호출하면서 인수로 7줄에서 만든 딕셔너리 데이터 세트를 전달합니다. dict_to_list 함수는 딕셔너리 데이터 세트의 각 요소를 [요소의 키, 요소의 값] 형식으로 리스트 데이터 세트로 변환한 값을 리턴합니다(실행 결과 1줄).

7.

```
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]
```

해설

- 1줄: count_down 함수를 정의합니다. 이 함수는 1개의 입력값을 받을 수 있고, 함수에 전달된 입력값은 매개변수 number에 저장됩니다.
- 2줄: 비어있는 리스트를 만들고 변수 result에 저장합니다.
- 3줄: 파이썬 range 명령어(함수)를 사용해서 0부터 number−1까지 숫자 데이터를 저장한 레인지 데이터 세트를 만들고, 이 레인지 데이터 세트에 저장된 숫자 데이터 개수만큼 for 반복문을 사용해서 4줄을 반복 처리합니다. 이때 변수 x에는 매회 반복 처리 시 0부터 number−1까지 숫자가 차례대로 저장됩니다.
- 4줄: 리스트 데이터 세트 명령어 append를 사용해서 result 리스트 끝에 (number − x)를 계산한 결괏값을 저장합니다. 예를 들어 number가 10인 경우, 첫 번째 반복 처리 시에는 x에 저장된 값이 0이므로, (10 − 0)을 계산한 결괏값을 result에 추가합니다.

```
result.append(number - x)
→ result.append(10 - 0)
→ result.append(10)
```

- number가 10인 경우 매회 반복 처리 시마다 number, x, result에 저장된 값의 변화를 표로 나타내면 다음과 같습니다.

반복 횟수	number	x	number − x	result.append()	result
1	10	0	10	result.append(10)	[10]
2	10	1	9	result.append(9)	[10, 9]
3	10	2	8	result.append(8)	[10, 9, 8]
4	10	3	7	result.append(7)	[10, 9, 8, 7]
5	10	4	6	result.append(6)	[10, 9, 8, 7, 6]
6	10	5	5	result.append(5)	[10, 9, 8, 7, 6, 5]
7	10	6	4	result.append(4)	[10, 9, 8, 7, 6, 5, 4]
8	10	7	3	result.append(3)	[10, 9, 8, 7, 6, 5, 4, 3]
9	10	8	2	result.append(2)	[10, 9, 8, 7, 6, 5, 4, 3, 2]
10	10	9	1	result.append(1)	[10, 9, 8, 7, 6, 5, 4, 3, 2, 1]

• 5줄: 2~4줄의 코드 처리 후 result에 저장된 리스트를 리턴합니다.

• 7줄: cound_down 함수를 호출하면서 10을 입력값으로 전달한 뒤, 그 리턴값(함수 호출 결괏값)을 화면에 출력합니다. 만약 10을 입력값으로 전달하면 위 표와 같이 [10, 9, 8, 7, 6, 5, 4, 3, 2, 1]을 화면에 출력합니다.

06-2 다양한 함수의 사례

1.

```
True
False
```

해설

• 1줄: 함수 is_odd는 함수 호출 시 전달된 입력값이 홀수(odd)인지 여부를 판단하고, 그 결괏값을 리턴하는 함수입니다.

• 2~3줄: 나머지 연산자(%)를 사용해서 매개변수 arg에 저장된 입력값을 2로 나눈 나머지를 구하고(arg % 2), 그 값이 1과 같으면(입력값이 홀수라면) 불 데이터 True를 리턴합니다. 한편, 3줄의 return 명령어가 실행되면 True를 리턴하면서 함수를 종료합니다. 즉 4줄은 실행되지 않습니다. 이러한 리턴 방식을 '조기 리턴'이라고 합니다.

• 4줄: 숫자 데이터를 2로 나눈 나머지는 0 또는 1인데, 나머지가 1인 경우(홀수)는 2~3줄에 의해 처리되고, 나머지가 0인 데이터(짝수)는 4줄에서 처리됩니다. 따라서 4줄에서는 False를 리턴합니다.

• 6줄: 함수 is_odd를 호출하면서 입력값으로 3을 전달합니다. 3은 홀수이므로, 함수의 리턴값으로 True를 얻을 수 있고, 이 값을 화면에 출력합니다(실행 결과 1줄).

• 7줄: 2는 짝수이므로, 함수의 리턴값으로 False를 얻을 수 있고, 이 값을 화면에 출력합니다(실행 결과 2줄).

2.

```
55
5050
```

해설

- 1줄: 함수 get_sum_of_two_numbers는 start부터 end까지 합계를 구하고, 그 값을 리턴하는 함수입니다.
- 2줄: 합계를 저장할 변수 result를 만들고, 초깃값으로 0을 저장합니다.
- 3~4줄: 파이썬 내장함수 range를 이용해서 start부터 end+1까지 레인지 데이터 세트를 만들고, for 구조를 사용해서 각 요소를 반복 처리합니다.
- 4줄: 합계를 저장할 result에 x(레인지 데이터 세트의 각 요소가 저장됨)를 더합니다. 이 명령어는 모든 레인지 데이터 세트의 요소에 반복 적용합니다.
- 5줄: 합계를 저장한 result를 리턴합니다.
- 7줄: 함수 호출 시 1과 10을 입력값으로 전달하고, 1부터 10까지 합계를 결괏값으로 얻을 수 있습니다. 이 결괏값을 화면에 출력합니다(실행 결과 1줄).
- 8줄: 함수 호출 시 1과 100을 입력값으로 전달하고, 1부터 100까지 합계를 결괏값으로 얻을 수 있습니다(실행 결과 2줄).

3.

```
#
##
###
```

해설

- 1줄: 함수 print_hashes를 정의합니다. 함수의 입력값으로 전달된 데이터는 매개변수 rows에 저장되고, rows는 함수 보디 2~3줄에서 사용할 수 있습니다.
- 2줄: for 반복문을 사용해서 rows에 저장된 숫자만큼 3줄을 반복 처리합니다. 만약 함수 입력값으로 3이 전달된 경우, range(3) 코드는 0부터 1씩 증가하는 숫자 3개를 저장한 레인지 데이

터 세트를 만들기 때문에, 변수 x는 매회 반복 처리할 때마다 0, 1, 2를 저장합니다.

- 3줄: x에 1을 더한 값만큼 "#"를 반복 연결한 결괏값을 화면에 출력합니다.
- 5줄: print_hashes 함수를 실행하는 코드입니다. 입력값으로 전달한 3은 1줄의 매개변수 rows에 저장됩니다.

4.

```
   #
  ##
 ###
####
```

해설

- 1줄: 함수 print_reverse_hashes를 정의합니다. 함수의 입력값으로 전달된 데이터는 매개변수 rows에 저장되고, rows는 함수 보디 2~4줄에서 사용할 수 있습니다.
- 2줄: for 반복문을 사용해서 rows에 저장된 숫자만큼 3~4줄을 반복 처리합니다. 만약 함수 입력값으로 4가 전달된 경우, range(4) 코드는 0부터 1씩 증가하는 숫자 4개를 저장한 레인지 데이터 세트를 만들기 때문에, 변수 x는 매회 반복 처리할 때마다 0, 1, 2, 3을 저장합니다.
- 3줄: 변수 x에 저장된 값에 1을 더한 값을 count 변수에 저장합니다. count 변수는 현재 실행 횟수를 1부터 시작하도록 만드는 데 사용합니다.
- 4줄: 현재 실행 횟수(count)만큼 "#" 문자를 반복 연결한 결괏값을 화면에 출력합니다. 다만 매회 반복 출력할 때마다 전체 실행 횟수(rows)에서 현재 실행 횟수(count) 값을 차감한 만큼 빈칸을 출력해서, 위 3번 문제의 결괏값을 세로로 뒤집은 모양을 출력합니다.
- 6줄: print_reverse_hashes 함수를 실행하는 코드입니다. 입력값으로 전달한 4는 1줄의 매개변수 rows에 저장됩니다.

정답 및 해설

5.

```
010-1234-****
010-9876-****
```

해설

- 1줄: 함수 replace_digits는 주어진 전화번호의 마지막 4자리를 별표로 변환한 값을 리턴하는 함수입니다.
- 2줄: 마지막 4자리 번호를 제거하기 위해서, 매개변수 str_에 저장된 함수의 입력값에 슬라이싱 [:9]를 적용하고, 마지막에 "****"를 연결해서 리턴합니다.
- 4줄: 함수를 호출하면서 문자 데이터 "010-1234-5678"을 전달합니다. 그 결괏값으로 "010-1234-****"를 얻을 수 있습니다(실행 결과 1줄).
- 5줄: 함수를 호출하면서 문자 데이터 "010-9876-5432"를 전달합니다. 그 결괏값으로 "010-9876-****"를 얻을 수 있습니다(실행 결과 2줄).