2
함수 제작 및 활용
Function Abstraction & Application

이 장에서는 시간단위를 변환하는 간단한 소프트웨어를 만들어보기로 한다. 이 과정에서 시큐어코딩의 개념과 조건문, 반복문, 함수 정의/호출의 기본 개념을 익힌다.

1. [사례 학습] 시간단위 변환

일상 생활에 많이 쓰는 시간의 단위로 년/해year, 월/달month, 일/날day, 시hour, 분minute, 초second가 있다. 한해는 열두달이고, 한달은 28~31날이고, 하루는 24시간이고, 1시간은 60분이고, 1분은 60초이다. 시간의 단위를 바꾸는 공식은 간단하지만 암산으로 하기는 복잡하고 일일이 손으로 적어서 계산하기도 귀찮으므로 컴퓨터의 도움을 받으면 좋겠다.

예를 들어, 하루가 몇 분인지 계산하여 출력하는 프로그램은 다음과 같이 작성할 수 있다.

```
1 print(1 * 24 * 60)
```

닷새가 몇 분인지 계산하여 출력하는 프로그램은 다음과 같이 작성할 수 있다.

```
2 print(5 * 24 * 60)
```

위의 두 줄을 편집기로 하나의 파일에 작성하여 파일이름을 `timetable.py`로 하여 저장하고 실행하여 결과를 확인해보자.

그런데 며칠을 분으로 환산할지 사용자가 임의로 정하게 하고 이를 분으로 환산해주면 효용성이 더 좋을 것이다. 다음과 같이 작성할 수 있다.

```
1 days = input('Enter a number : ‘ )
2 days = int(days)
3 print(days * 24 * 60)
```

`input()` 함수는 사용자 입력을 실행창에서 받아 문자열로 내주므로 곱셈을 하기 전에 `int`를 써서 정수로 변환해주었다. 기대한 대로 작동하는지 시험해보자. (입력 사례: `5`, `-7`, `+3`, `0`)
잠깐! 정수로 변환하지 않고 계산하면 어떤 결과가 나오는지 직접 실행해보고, 왜 그런 결과가 나오는지 생각해보자.

날짜를 분으로 환산해주는 프로그램을 만들기로 했다면 이 프로그램은 제 기능을 하는 프로그램이라고 할 수 있다. 그런데 프로그램을 작성하기 전, 프로그램이 만족시켜야 할 요구사항을 먼저 정하는 것이 원칙이다. 그러니 좀 늦었지만 날짜를 분으로 환산해주는 프로그램의 요구사항을 기술해보자.

날짜-분 환산 프로그램 : 요구사항 Requirements	
날짜day를 분minute으로 환산한다. 입출력은 실행창에서 한다.	
입력	정수
출력	입력을 분으로 환산한 정수 단, 입력이 음수인 경우에는 0

요구사항으로 어떤(날짜를 분으로 환산) 기능을 수행하는지 기술하고, 입력과 출력은 어떤 방식(실행창)으로 하는지 기술하고 있다. 그리고 입력으로 어떤 형식(정수)의 데이터를 허용하는지, 출력은 어떻게 하는지도 따로 기술하고 있다. 입력은 날짜를 나타내므로 정수만 허용하고, 출력은 분으로 환산한 정수로 표현하기로 정했다. 위에서 작성한 프로그램은 음수 입력도 정상적으로 처리한다. 그러나 시간 단위를 환산하는데 음수계산은 별 의미가 없으므로 음수 입력은 크기에 관계없이 0을 출력하라고 요구사항을 설정하였다.

그러면 이제 위의 날짜-분 환산 프로그램 요구사항에 맞추어 코딩하는 과정을 경험해보자. 이미 작성한 다음 프로그램이 위의 기능 요구사항을 만족하는지 우선 점검해보자.

```
1 days = input('Enter a number : ‘ )
2 days = int(days)
3 print(days * 24 * 60)
```

이 프로그램에 정수를 입력하면 기대한 대로 맞는 답을 환산해주면서 프로그램이 정상적으로 종료한다. 위에서 시험해본 대로 이 프로그램은 `5`, `-7`, `+3`, `0` 과 같은 입력에 대해서 문제없이 환산 결과를 출력해 주었다.

그런데 요구사항을 보면 0 이상의 정수만 환산해주고, 음수의 경우 0을 출력하라고 한다.
이 새로운 요구사항에 맞게 기능이 작동하게 하려면 프로그램을 다음과 같이 수정하면 된다.

```
1 days = input('Enter a number : ‘ )
2 days = int(days)
3 if days >= 0 :
4     print(days * 24 * 60)
5 else :
6     print(0)
```

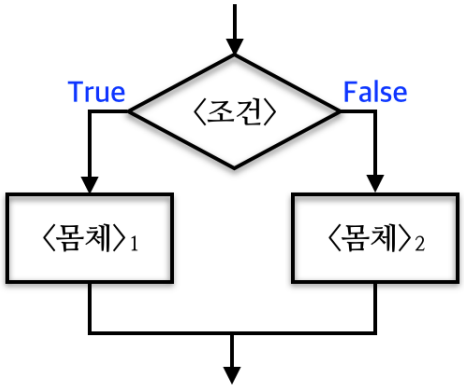
수정한 프로그램을 위와 같은 입력사례로 실행하여 어떤 결과가 나오는지 관찰해보자. 0 이상의 정수 입력에 대해서는 이전 프로그램과 동일하게 작동한다. 그리고 음의 정수를 입력하면 0을 프린트한다. 이제 요구사항에서 제시한 기능을 충실하게 수행하는 프로그램을 완성하였다.

어떻게 이런 결과가 나왔을지 이 개선한 프로그램을 읽어보고 실행의미를 추측해보자. 여기서 `if` 와 `else` 같은 단어는 생소할텐데, 이와 같은 단어가 포함된 구문을 조건문이라고 한다. 그러면 이제 조건문의 문법과 의미를 공부해보자.

조건문

조건문conditional command의 **문법**syntax/grammar은 다음과 같다.

```
if 〈조건〉 :
    〈몸체〉1
else :
    〈몸체〉2
```



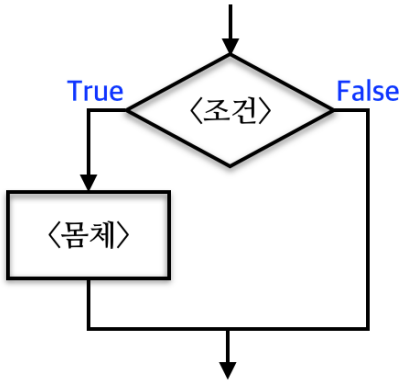
〈조건〉은 논리식이며, 〈몸체〉에는 한 줄 이상의 명령문command을 나열할 수 있다. 지금까지 배운 명령문으로는 지정문과 프린트문, 조건문이 있다.

조건문의 **의미**semantics/meaning는 위 오른쪽 그림과 같이 **흐름도**flow chart로 표현할 수 있다. 〈조건〉의 논리식을 계산한 결과가 `True`이면 〈몸체〉1을 실행하고, `False`이면 〈몸체〉2을 실행한다.

조건문의 〈몸체〉는 반드시 일정하게 **들여쓰기**indentation를 해야하는데, 같은 간격으로 들여쓴 명령문 덩어리 전체를 블록block이라 한다. 일반적으로 Python은 4칸씩 들여쓰기를 권장하므로 우리도 그렇게 하기로 하자. 들여쓰기는 일정하게 잘 맞추어야 한다. 즉, 〈몸체〉1과 〈몸체〉2는 같은 간격으로 들여쓰기를 해야한다. 들여쓰기가 일정하지 않으면 구문오류로 처리된다. 바로 위의 사례 코드에서 조건문 내의 두 블록이 각각 4칸씩 들여쓰기가 되어있음을 확인해보자. Python 커뮤니티는 4칸씩 들여쓰기를 관습적으로 있으므로 그렇게 하기를 권장한다. Python 전용 편집기를 사용하면 자동으로 들여쓰기를 해줘서 편리하다.

조건문의 `else` 이하는 생략해도 된다.

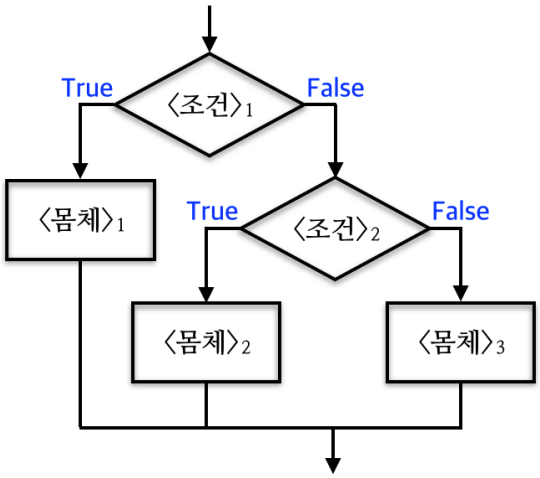
```
if 〈조건〉 :
    〈몸체〉
```



`else` 가 생략된 조건문의 의미는 오른쪽의 흐름도와 같다.

조건이 여럿 있고 조건에 따라서 다르게 계산해야하는 경우, 아래와 같이 `elif` 절을 추가하면 된다. `elif` 절은 필요한대로 많이 써도 된다. 여기서도 마지막 `else` 절은 생략할 수 있다.

```
if 〈조건식〉1 :
    〈몸체〉1
elif 〈조건식〉2 :
    〈몸체〉2
else :
    〈몸체〉3
```



〈조건식〉1의 계산결과가 `True`이면 〈몸체〉1을 실행하고, 〈조건식〉1의 계산결과가 `False`이고 〈조건식〉2의 계산결

과가 True이면 〈몸체〉$_2$을 실행하고, 〈조건식〉$_2$의 계산결과도 False이면 〈몸체〉$_3$을 실행한다.

조건문 이해 확인용 프로그램 예제

이제 조건문을 이해했는지 확인해보자. 다음 프로그램을 먼저 눈으로 읽어서 어떤 프로그램인지 이해하고, 직접 실행하여 제대로 이해했는지 확인해보자.

```
 1 score = input('Enter your score : ')
 2 score = int(score)
 3 if 90 <= score <= 100:
 4     print('A')
 5 elif 80 <= score <= 89:
 6     print('B')
 7 elif 70 <= score <= 79:
 8     print('C')
 9 elif 60 <= score <= 69:
10     print('D')
11 elif 0 <= score <= 59:
12     print('F')
13 else:
14     print('Invalid input!')
```

날짜를 분으로 환산하는 프로그램을 다시 한번 음미해보자.

```
1 days = input('Enter a number : ‘ )
2 days = int(days)
3 if days >= 0 :
4     print(days * 24 * 60)
5 else :
6     print(0)
```

이 프로그램은 정수 입력에 대해서 완벽히 작동하도록 작성하였다. 즉, 0 이상의 정수에 대해서는 날짜를 분으로 환산을 해주고, 0 미만의 정수에 대해서는 0을 출력해주도록 하였다. 즉, 입력이 정수로 주어지는 한 정상적으로 작동하는 프로그램이다. 그러나 정수가 아닌 입력에 대해서는 ValueError 오류메시지와 함께 비정상적으로 종료한다. 실행창에서 직접 확인해보자.

프로그램 사용자가 안내하는대로 착하게 입력할 것이라 단정하는 건 금물이다. 차라리 사용자는 모두 내 프로그램을 오작동 시키려고 작정한 해커라고 생각하고 프로그램을 만드는 것이 안전하다. 따라서 사용자 입력은 예외없이 모두 **입력확인**(input validation)하여 불량입력이 프로그램에 투입되는 걸 막아야 한다. 즉, 사용자 입력이 프로그램에서 기대하고 있는 부류인지 반드시 확인하여 그렇지 않은 경우 걸러내거나 다시 입력 받도록 프로그램을 짜야한다. 입력확인을 제대로 하지 않아 잘못된 사용자 입력이 프로그램 실행과정에 투입된다면 의도하지 않았던 결과를 초래할 수 있다. 사실 악의적인 해킹이나 소프트웨어의 치명적인 오류는 사용자 입력을 제대로 확인하지 않아 발생하는 경우가 대부분이다. 따라서

입력을 반드시 확인하여 어떤 기형 불량입력에 대해서도 프로그램이 비정상 종료하지 않고 버틸 수 있도록 코딩해야 한다. 이를 **시큐어코딩**(secure coding)이라고 한다.

시큐어코딩을 보장하기 위해서는 이를 요구사항에 명시해야 한다. 앞에서 작성한 요구사항은 기능에 촛점을 맞추었다면, 이와는 별도로 보안에 촛점을 맞춘 보안 요구사항이 별로로 제시되어야 한다. 앞에서 작성한 기능 요구사항에 보안 요구사항을 추가하여 다음과 같이 요구사항을 작성하였다.

날짜-분 환산 프로그램 : 요구사항 Requirements		
기능	날짜day를 분minute으로 환산한다. 입출력은 실행창에서 한다.	
	입력	정수
	출력	입력을 분으로 환산한 정수 단, 입력이 음수인 경우에는 0
보안	0 이상의 정수가 아닌 입력은 재입력을 받도록 조치한다.	

새로 추가된 보안 요구사항은 0 이상의 정수가 아닌 입력은 재입력 받도록 하라는 것이다. 이 요구사항을 충족하기 위해서는 사용자 입력을 검사하여 입력에 숫자()가 아닌 문자가 포함되어 있으면 불량입력으로 취급하여 무시하고 다시 입력하도록 요청하면 된다. 이제 어떻게 이 보안요구사항을 구현하는지 공부해보자.

입력확인

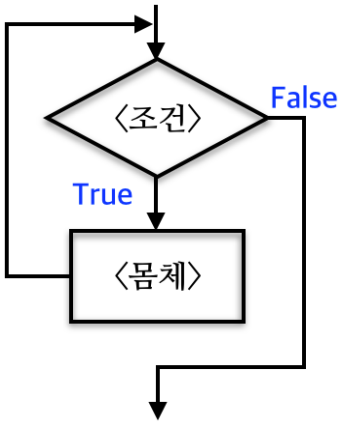
위 프로그램에서 0~9의 숫자가 아닌 다른 문자로 포함된 입력이 들어오는 경우, 무시하고 사용자에게 다시 입력을 요청하도록 프로그램을 다음과 같이 수정할 수 있다. 사용자 입력을 받아서 정수로 바꾸기 전 입력을 확인하도록 했는데 진하게 표시된 부분이다.

```
1 days = input('Enter a number : ‘ )
2 while not days.isdigit():
3     days = input('Enter a number : ‘ )
4 days = int(days)
5 if days >= 0 :
6     print(days * 24 * 60)
7 else :
8     print(0)
```

while 문을 처음 만났으니 while 문의 문법과 의미를 먼저 공부한 뒤에 돌아와서 이 프로그램을 이해해보도록 하자.

while 반복문

while 반복문은 지정한 조건이 만족하는 동안 동일 블록을 반복 실행하도록 고안한 프로그램 구조이다.



반복문iteration command의 문법은 다음과 같다.

```
while 〈조건〉 :
    〈몸체〉
```

〈조건〉은 논리식이며, 〈몸체〉는 한 줄 이상의 명령문으로 구성된다. 〈몸체〉는 지금까지 배운 명령문인 지정문과 프린트문, 조건문 뿐만 아니라 반복문까지도 포함할 수 있다. 반복 실행의 대상인 〈몸체〉 부분은 조건문과 마찬가지로 네 칸 들여쓰기를 하여 줄을 맞추어야 하나의 블록으로 인식한다.

이 반복문의 의미는 위의 오른쪽 흐름도로 이해하면 제일 쉽다. 말로 설명하면, 〈조건〉의 계산 결과가 True인 동안 〈몸체〉을 반복 실행하다가, 〈조건식〉의 계산 결과가 False가 되면 반복문을 빠져나와 다음으로 넘어간다.

입력확인 (계속)

이제 while 문을 이해했으니 이 프로그램의 의미를 이해해보자.

```
1 days = input('Enter a number : ' )
2 while not days.isdigit():
3     days = input('Enter a number : ' )
4 days = int(days)
5 if days >= 0 :
6     print(days * 24 * 60)
7 else :
8     print(0)
```

줄 1에서 days 변수에는 입력받은 문자열이 지정된다. 줄 2~3는 반복문이다. 조건식인 not days.isdigit() 의 계산 결과가 True가 되는 한, 몸체인 줄 3을 반복 실행하여 사용자로부터 재입력을 받는다. 다시 말해, not days.isdigit() 이 False가 될 때까지, 또 다른 말로 days.isdigit() 이 True가 될 때까지, 계속 재입력을 받는다. days.isdigit() 가 통과조건인 셈이다. days.isdigit() 이 생소하므로 따로 공부하여 이해한 다음 다시 돌아오자.

문자열 메소드 : isdigit()

Python 언어 공식사이트에서는 바로 가져다 쓸 수 있는 부품 라이브러리 목록을 자세히 기록한 문서인 The Python Standard Library 문서[1]를 제공한다. 그 문서 4.7.1 String Methods 를 뒤져보면 문자열을

[1] The Python Standard Library 문서는 https://docs.python.org/2/library/index.html 에서 참고

처리하는데 쓸 수 있는 유용한 **메소드**method를 많이 찾을 수 있다. (메소드는 문자열이 수행할 수 있는 기능으로 이해하면 된다.) 이 중 하나가 `isdigit()` 이다.

문자열 메소드는 다음과 같은 형식으로 호출한다.

〈문자열〉.isdigit()

이 호출은 〈문자열〉을 계산한 문자열이 모두 숫자이면 `True`, 그렇지 않고 하나라도 숫자가 아닌 문자가 있으면 `False`를 내준다. 다음 예를 하나씩 실행창에서 확인하면서 의미를 이해해보자.

```
"345".isdigit()
"-45".isdigit()
"4.5".isdigit()
"4 7".isdigit()
```

사례 : 날짜-분 변환 (입력확인 계속)

이제 통과조건인 `days.isdigit()`을 이해하였다.

```
1 days = input('Enter a number : ‘ )
2 while not days.isdigit():
3     days = input('Enter a number : ‘ )
4 days = int(days)
5 if days >= 0 :
6     print(days * 24 * 60)
7 else :
8     print(0)
```

사용자가 입력한 문자열이 모두 숫자로만 구성되어 있어야 통과조건을 만족한다. 입력 문자열이 숫자로만 구성되어 있으면 이 조건식이 `False`가 되어 몸체(줄 3)를 실행하지 않고 넘어가지만, 하나라도 숫자가 아닌 문자가 포함되어 있으면 조건식이 `True`가 되어 몸체를 실행하여 다시 입력을 받는다. 따라서 이 조건을 만족하지 않는 한 사용자는 다시 입력을 할 수밖에 없게 프로그램이 작동한다.

`while` 문을 빠져나와 줄 4에 도달하면 `days` 변수의 값은 숫자로만 구성된 문자열이 확실하다. 이제는 `int`로 타입변환을 해도 오류가 절대 발생하지 않을테므로 안전하다. 줄 4의 지정문을 실행하면 `days` 변수의 값은 정수로 바뀌어 재지정된다.

그런데 입력확인을 거친 입력은 모두 숫자로만 구성되어 있으므로, 절대로 음수가 될 수 없다. 따라서 `days` 변수 값이 `0` 이상인지를 굳이 검사할 필요가 없어졌다. 이 조건문을 지우면 보안 요구사항까지 만족하는 프로그램이 다음과 같이 완성된다.

```
1 days = input('Enter a number : ‘ )
2 while not days.isdigit():
3     days = input('Enter a number : ‘ )
4 days = int(days)
5 print(days * 24 * 60)
```

그런데 이 프로그램은 사용자의 편의성을 높여주는 설명이 전혀 없다. `print` 명령문을 적절히 다음과 같이 삽입하여 사용자 편의를 높일 수 있다.

```
print("날짜를 분으로 환산해드립니다.")
days = input('며칠? ')
while not days.isdigit():
    print("숫자가 아닌 문자가 들어있습니다.")
    days = input('며칠?: ')
days = int(days)
print(days, "일은", days * 24 * 60, "분입니다.")
print("이용해주셔서 감사합니다.")
```

이 프로그램을 다음의 입력 사례로 각각 실행하고 결과를 관찰하여, 절대 오류가 발생하지 않는 안전한 프로그램임을 확인해보자.

사흘
닷새
-9
+3
0
5

입력 확인 패턴

앞에서 보았듯이 사용자 입력확인을 하는 코드 패턴은 다음과 같다.

```
s = input()
while not 〈통과조건〉 :
    s = input()
```

통과시킬 우량 입력이 될 조건을 입력문자열 s를 이용하여 논리식으로 만들어 〈통과조건〉의 위치에 넣으면 된다. 앞으로 많이 써야 할테니 패턴을 익혀두자. 앞에서 공부한 날짜-분 변환 프로그램에서는 숫자로 구성된 문자열만 통과시켜야 하므로 통과조건이 `s.isdigit()` 이었음을 기억하자.

서비스의 무한 반복

위 프로그램은 한번 사용하면 프로그램이 끝나버린다. `while` 반복문을 활용하여 필요한 만큼 얼마든지 사용할 수 있도록 할 수 있다. 반복하려는 코드 〈블록〉을 다음과 같이 `while True:` 로 감싸면 블록 몸체를 무한정 반복하여 실행한다.

```
while True:
    〈몸체〉
```

반복을 그만하고 바깥으로 나가고 싶으면 `break` 명령을 사용할 수 있다. 〈몸체〉 안에서 `break` 명령을 실행하면 `break` 명령을 포함하고 있는 블록을 감싸고 있는 가장 가까운 `while` 문의 바깥으로 실행 흐름이 빠져나간다. (이렇게 얘기해야 하는 이유는 반복문의 몸체 내부에 또 반복문이 있을 수 있기 때문이다.)

다음 프로그램은 날짜를 분으로 환산하는 계산을 사용자가 원하는 한 계속하다, 사용자가 그만하기를 원하는 경우 프로그램을 멈추도록 작성하였다. 바깥의 while 문(줄 2~13)의 몸체 블록은 줄 3~13이며 블록 내부에 있는 break 명령이 실행되지 않는 한 이 몸체 블록은 무한정 반복 실행된다.

```
1  print("날짜를 분으로 환산해드립니다.")
2  while True:
3      days = input('며칠? ')
4      while not days.isdigit() :
5          print("숫자가 아닌 문자가 들어있습니다.")
6          days = input('며칠? ')
7      days = int(days)
8      print(days, "일은", days * 24 * 60, "분입니다.")
9      cont = input( " 계속하시겠습니까?(예/아니오) ")
10     while not (cont == '예' or cont == '아니오'):
11         cont = input("계속하시겠습니까?(예/아니오) ")
12     if (cont == '아니오'):
13         break
14 print("이용해주셔서 감사합니다.")
```

몸체 블록 끝 부분의 진하게 표시한 추가 코드(줄 9~13)의 의미를 잘 새겨보자. 줄 9~11은 사용자로 부터 '예' 또는 '아니오' 입력을 받아서 cont 변수로 지정한다. 줄 12에 도달하면 cont 변수가 가지고 있는 문자열은 '예' 또는 '아니오' 이다. '아니오'인 경우 break 문을 실행하게 되어 while 문을 빠져나와 줄 14번으로 넘어가지만, 그렇지 않으면 줄 2번으로 돌아간다. 여기서 break 문을 감싸고 있는 while 문은 2번이지 10번이 아님을 확인하고 넘어가자. 사용자가 원하는 동안 특정한 일을 무한정 반복하고 싶은 경우 사용하는 전형적인 패턴이므로 익혀두도록 하자.

요구사항에 맞추어 프로그램을 작성하는 작업을 **코딩**coding이라고 하는데, 다음과 같이 두 단계로 나누어 차례로 진행하는 것이 좋다.

- **기능우선코딩**functional coding : 기능 요구사항에 명시한 기능의 작동에 촛점을 맞추어 프로그램을 작성하는 코딩작업을 말한다. 기능이 잘 작동하는지 확인해줄 우량입력에 대해서만 프로그램이 완벽하게 작동하면 충분하다.
- **시큐어코딩**secure coding : 임의의 불량입력에 대해서 프로그램이 잘 대응하여 기능 작동의 안전을 보장하는 코딩작업을 말한다. 불량입력은 걸러내고 우량입력만 통과하는 **입력확인**input validation 필터를 추가하여, 기대하지 않은 불량입력은 절대 유입되지 못하도록 해야한다.

2. 함수 (= Function Abstraction)

컴퓨터를 분해해보면 여러 종류의 작은 부품들이 결합되어 만들었음을 알 수 있다. 이 뿐만 아니라 우리 주위에 있는 대부분의 물건들은 작은 부품들을 적절히 잘 골라서 조립하여 만들어진 완성품이다. 많이 쓰이는 부품은 표준을 만들어 규격화하여 대량생산 해놓고 필요에 맞게 골라 쓸 수 있게 하기도 하고, 특정 용도로만 사용되는 부품은 자가 제작하여 쓰기도 한다.

소프트웨어를 구성하고 있는 프로그램도 마찬가지다. 특정한 일을 하는 프로그램을 부품으로 만들어 이름을 붙여놓고 필요할 때마다 불러서 가져다 쓸 수 있으면 좋다. 소프트웨어가 복잡해 질수록 더욱 더 부품 제작이 절실해 진다. 자주 쓸 만한 부품들은 미리 만들어 모아놓고 필요할 때 가져다 쓰면 편리할 것이다. 이미 공부한 붙박이 함수가 그런 것이다.

라이브러리 붙박이 함수

지금까지 공부하면서 언제든지 불러쓸 수 있도록 미리 만들어 놓은 **붙박이함수**built-in function를 몇 개 써봤다.

- 제일 먼저 써본 붙박이 함수는 `print` 함수였다. 이 함수는 뒤에 나열된 하나 이상의 표현식을 차례로 계산하고 그 결과를 실행창에 차례로 출력하여 보여주는 역할을 하였다. 이 함수는 사실 프린트만 할 뿐, 내주는 값은 없다. 이와 같이 내주는 값이 없는 함수를 **프로시저**(procedure)라고 하기도 한다. 실행창에 프린트하는 행위는 함수의 실행 결과가 아니고 함수의 몸체를 실행하는 과정에서 생겨난 **부수효과**(side effect)일 뿐이다.
- 다음에 써본 붙박이 함수는 `input` 함수였다. 이 함수는 실행창에서 사용자의 입력을 기다린다. 사용자가 입력을 하고 Enter 키를 누르면 입력한 문자열을 결과로 내준다.
- `int`, `float`, `str` 과 같은 타입변환 함수도 써봤다. 각 함수는 가능한 경우 요구하는 타입으로 변환한 값을 내주는 함수이다.

Python에 어떤 붙박이 함수가 있는지 공식사이트의 **표준 라이브러리**(The Python Standard Library)의 2. Built-in Functions 에서 모두 찾아볼 수 있다.이 붙박이 함수들은 우리가 언제나 불러 쓸 수 있는 기본 부품이라고 할 수 있다. 파이선 공식사이트를 방문하여 어떤 붙박이 함수가 있는지 구경해보자.

표준 라이브러리 문서 4.7.1 String Methods를 뒤져보면 `isdigit`과 같은 문자열을 다루는 다양한 라이브러리 **메소드**method가 기술되어 있다. 문자열을 다룰 때 유용하게 쓸 수 있는 메소드들이다. 메소드는 호출하는 방식이 함수와 다르지만 미리 만들어 놓은 붙박이라는 점은 같다.

자가제작 함수

어떤 경우에는 특정 용도로 맞춤형 부품을 별도로 자가제작하여 써야 할 때도 있다. 이를 **사용자정의 함수**user-defined function 라고 한다.

프로그램 안에 같은 코드가 여러번 중복 나타나면서 길어지면 프로그램을 이해하거나 유지관리하기 어려워진다. 코드의 일부분을 수정하는 경우 중복코드를 모두 찾아서 일관성 있게 수정해야 할 수도 있기 때문이다. 만약에 한 군데라도 빠트리게 되면 프로그램 오류를 야기하는 원인이 될 수 있다. 따라서

중복되는 부분은 함수로 만들어두고 필요할 때마다 불러쓰도록 하면 프로그램의 이해가 쉬워지고, 수정이 필요한 경우에 한번만 고치면되므로 유지관리가 쉬워진다.

함수 정의와 호출: 문법과 의미

함수의 작성 행위를 **함수 정의**function definition라고 하고, 불러쓰는 행위를 **함수 호출**function call이라고 한다.

함수를 정의하는 문법은 다음과 같다.

```
def 〈함수이름〉 ( 〈변수〉1, 〈변수〉2, … , 〈변수〉n ) :
    〈몸체〉
```

〈함수이름〉은 변수 이름을 만드는 규칙과 동일하다. 괄호 안에 들어가는 변수들은 필요한 대로 0개 이상 나열할 수 있는데 〈몸체〉에서 사용할 변수 이름을 지정하는 것으로 **형식파라미터**formal parameter라고 한다. 〈몸체〉는 1줄 이상의 명령문으로 구성된 블록으로 다른 블록과 마찬가지로 4칸 들여쓰기를 해야한다. 함수를 정의할 때 몸체는 실행하지 않는다. 함수에서 만들어 낸 정보를 결과로 내주려면 다음과 같은 형식으로 〈몸체〉의 끝부분에 `return` 문을 달아두어야 한다.

```
return 〈표현식〉
```

그러면 〈표현식〉을 계산한 결과를 함수 호출의 결과로 내준다. `return` 문이 없으면 함수 호출은 아무런 값도 내주지 않으며, `return` 문이 없는 함수를 **프로시저**procedure라고 한다.

함수를 호출하는 문법은 다음과 같다.

```
〈함수이름〉 ( 〈표현식〉1, 〈표현식〉2, … , 〈표현식〉n )
```

이미 정의되어 있는 함수만 호출이 가능하며, 괄호 안에 들어가는 표현식은 **실제파라미터**actual parameter 또는 **인수**argument라고 하며, 호출을 하면 인수(실제파라미터)를 계산한 결과 값을 각각 짝을 맞추어 형식파라미터의 변수에 지정을 하고 함수의 〈몸체〉를 실행한다. 따라서 일반적으로 실제파라미터(인수)의 개수와 형식파라미터의 개수는 일치해야 한다.

함수작성 연습문제

1. 반지름을 입력받아 원의 면적을 구하여 내주는 함수 `area_circle`을 정의해보자. 반지름이 r 이면 원의 면적은

$$\pi r^2$$

π값은 Python의 `math` 모듈에 `pi`라는 이름으로 정해 놓았으므로 가져다 쓰면 된다. 모듈을 가져다 쓰려면 다음과 같이 쓰기전에 수입import해야 한다.

```
>>> import math
>>> math.pi
3.141592653589793
```

원의 면적을 구하는 Python 표현식은 `math.pi * r ** 2` 이다.

이제 다음의 빈 공간을 채워서 area_circle 함수를 작성해보자. 계산 결과에서 소수점 둘째자리 미만은 반올림하도록 하자.

```
def area_circle(r):
    import math
    return
```

작성한 함수가 잘 작동하는지 다음 호출로 시험해보자.

```
area_circle(5)
area_circle(9)
area_circle(23)
```

힌트 붙박이 함수의 하나로 제공하고 있는 round를 사용하면 소수점 이하 자리수를 지정하여 반올림할 수 있다. 다음 예를 실행창에서 실행하여 예상대로 반올림을 하는지 관찰해보자.

```
round(1.3)
round(1.5)
round(2.5)
round(2.6)
```

4 이하이면 버리고 5 이상이면 올리는 것이 반올림인데, 위의 예의 경우 1.5은 2로 올리지만, 2.5는 소수점 이하를 버린다. 이와 같은 현상은 실수 오차로 발생하는 현상이므로 어쩔 수 없다. 소수점 특정 자리수 아래에서 반올림하고 싶으면 자리수를 추가로 다음과 같이 명시하면 된다.

```
round(2.356, 2)
```

이 경우 소숫점 2째 자리 아래를 반올림하여 결과는 2.36 이 된다. 이 경우에도 오차는 생길 수 있으니 감수해야 한다.

2. 1번 문제를 했으면, 이번엔 area_circle의 파라미터 k를 추가하여 반올림할 자리수를 인수로 전달할 수 있도록 프로그램을 수정해보자.

```
def area_circle(r,k):
    import math
    return
```

작성한 함수가 잘 작동하는지 다음 호출로 시험해보자.

```
area_circle(9,2)
area_circle(9,3)
area_circle(9,4)
```

3. [사례 학습] 시간단위 변환 (함수 만들기)

함수 만드는 방법을 알았으니 이를 활용하여 앞에서 작성한 다음 날짜-분 변환 프로그램을 개선해보자.

```
1  print("날짜를 분으로 환산해드립니다.")
2  while True:
3      days = input('며칠? ')
4      while not days.isdigit() :
5          print("숫자가 아닌 문자가 들어있습니다.")
6          days = input('며칠? ')
7      days = int(days)
8     print(days, "일 =", days * 24 * 60, "분")
9      cont = input("계속하시겠습니까?(예/아니오) ")
10     while not (cont == '예' or cont == '아니오'):
11         cont = input("계속하시겠습니까?(예/아니오) ")
12     if (cont == '아니오'):
13         break
14 print("이용해주셔서 감사합니다.")
```

위 프로그램에서 앞으로 함수로 만들어두면 활용할 수 있을 만한 부분 줄 3~7을 진하게 표시했다. 이 부분은 사용자로부터 정수 문자열을 입력받아 정수로 변환하는 일을 한다. 이 부분을 get_number 라는 이름의 함수로 정의하고 호출하여 쓰도록 수정하면 다음과 같다. 위 프로그램의 줄 3~7 부분이 아래 프로그램에서 정의한 get_number 함수의 몸체(줄 2~6)가 되고, 그 부분은 아래 프로그램의 줄 10과 같이 함수 호출로 대체하였다.

```
1  def get_number():
2      days = input('며칠? ')
3      while not days.isdigit() :
4          print("숫자가 아닌 문자가 들어있습니다.")
5          days = input('며칠? ')
6      return int(days)
7
8  print("날짜를 분으로 환산해드립니다.")
9  while True:
10     days = get_number()
11     print(days, "일 =", days * 24 * 60, "분")
12     cont = input("계속하시겠습니까?(예/아니오) ")
13     while not (cont == '예' or cont == '아니오'):
14         cont = input("계속하시겠습니까?(예/아니오) ")
15     if (cont == '아니오'):
16         break
17 print("이용해주셔서 감사합니다.")
```

줄 10에서 get_number() 함수를 호출하면 줄 2~6을 실행하여 사용자에게서 받은 정수 문자열을 정수로 바꾸어 리턴해주고, 호출한 그 값을 받아서 줄 10의 days 변수로 지정한다. 이 사례에서는 함수를 만

들어서 얻는 점이 무엇인지 의아해할 수 있지만, 사용자에게서 정수 입력을 여러번 받아야 하는 프로그램에서는 유용하게 써먹을 수 있다. 같은 코드를 중복하여 작성하는 대신 이 함수를 호출하면 되니 코드가 훨씬 간결해질 것이다.

위 프로그램에서 수를 입력받을 때 사용자에게 보여주는 메시기가 두 가지 있다. 하나는 '며칠? ' 이고, 다른 하나는 "숫자가 아닌 문자가 들어있습니다." 이다. 이 함수를 수를 입력받을 용도로 다양하게 사용할 수 있으려면, 이 메시지도 용도에 맞게 다양하게 제공할 수 있으면 좋겠다. 이 메시지를 호출할 때 인수로 전달해주도록 하면 이 함수를 더 유연하게 활용할 수 있다. 메시지를 인수로 전달하도록 함수 정의와 호출를 수정하면 다음과 같다.

```
1  def get_number(message1,message2):
2      days = input(message1)
3      while not days.isdigit() :
4          print(message2)
5          days = input(message1)
6      return int(days)
7
8  print("날짜를 분으로 환산해드립니다.")
9  while True:
10     days = get_number('며칠? ',"숫자가 아닌 문자가 들어있습니다.")
11     print(days, "일 =", days * 24 * 60, "분")
12     cont = input("계속하시겠습니까?(예/아니오) ")
13     while not (cont == '예' or cont == '아니오'):
14         cont = input("계속하시겠습니까?(예/아니오) ")
15     if (cont == '아니오'):
16         break
17 print("이용해주셔서 감사합니다.")
```

메시지로 사용할 문자열은 함수 호출의 인수로 전달하고, 전달 받은 문자열은 각각 함수 파라미터 변수에 지정되어 함수 본체에서 사용된다.

바로 위 프로그램의 줄 12~15는 사용자로 부터 계속할지말지를 물어보는 일을 하는 부분인데, 다음과 같이 stop() 함수로 만들어 호출해쓰면 좋겠다. 이 함수는 사용자에게 계속할지말지 여부를 물어본 뒤에 그 여부를 논리값으로 리턴한다. 그러면 반복문을 빠져나갈지 말지를 결정하는 조건식에서 이 함수를 호출할 수 있다. 아래 프로그램의 줄 8~12 부분이 새로 정의한 stop 함수이다. 위 프로그램의 줄 12~15 부분이 아래 프로그램에서 정의한 stop 함수의 몸체(줄 9~12)가 된다. 이 함수는 사용자가 '아니오'를 입력하면 True 값을 리턴해주고, '예'를 입력하면 False 값을 리턴해준다. 그러므로 아래 프로그램의 줄 18의 조건에서 이 함수를 호출한다.

```
1  def get_number(message1,message2):
2      days = input(message1)
3      while not days.isdigit() :
4          print(message2)
5          days = input(message1)
6      return int(days)
7
8  def stop():
9      cont = input("계속하시겠습니까?(예/아니오) ")
10     while not (cont == '예' or cont == '아니오'):
11         cont = input("계속하시겠습니까?(예/아니오) “ )
12     return cont == '아니오'
13
14 print("날짜를 분으로 환산해드립니다.")
15 while True:
16     days = get_number('며칠? ',"숫자가 아닌 문자가 들어있습니다.")
17     print(days, "일 =", days * 24 * 60, "분")
18     if stop():
19         break
20 print("이용해주셔서 감사합니다.")
```

이제 자가제작 함수를 2개 정의하였고 이를 활용한 프로그램은 줄 14~20 이다. 이 부분은 프로그램 지휘본부라고 할 수 있는데, 일반적으로 메인main 프로그램 라고 한다. 그런데 이마저도 함수로 만들어 두면 좋다. 이 경우는 아래와 같이 전체를 함수로 정의하고 호출하면 되겠다. 사실 이 부분은 값을 리턴할 일이 없기 때문에 return 문을 둘 필요가 없다. 앞에서 언급했지만 리턴문이 없는 함수를 프로시저라고 한다. 프로시저의 정의는 단순히 명령문 블록에 이름붙인 걸로 이해하면 되고, 프로시저를 호출해야 비로서 몸체의 명령문을 실행하게 된다.

```
1  def get_number(message1,message2):
2      days = input(message1)
3      while not days.isdigit() :
4          print(message2)
5          days = input(message1)
6      return int(days)
7
8  def stop():
9      cont = input("계속하시겠습니까?(예/아니오) ")
10     while not (cont == '예' or cont == '아니오'):
11         cont = input("계속하시겠습니까?(예/아니오) “ )
12     return cont == '아니오'
13
14 def day2minute():
15     print("날짜를 분으로 환산해드립니다.")
16     while True:
17         days = get_number('며칠? ',"숫자가 아닌 문자가 들어있습니다.")
18         print(days, "일 =", days * 24 * 60, "분")
19         if stop():
20             break
21     print("이용해주셔서 감사합니다.")
22
23 day2minute()
```

1절에서 만들 프로그램을 3개의 함수를 만들어 재구성해 보았다. 이 프로그램을 실행하기 위하여 줄 23과 같이 프로그램의 지휘본부 역할을 하는 메일 프로시저인 day2minute()를 호출해야 한다.