10장 내장 함수와 모듈

내장 함수

- 내장 함수는 대부분의 객체에 대해서 사용이 가능하다.

abs()	dict()	help()	min()	setattr()
all()	dir()	hex()	next()	slice()
any()	divmod()	id()	object()	sorted()
ascii()	enumerate()	input()	oct()	staticmethod()
bin()	eval()	int()	open()	str()
bool()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	
delattr()	hash()	memoryview()	set()	

abs(x) 함수

```
>>> abs(-3)
3
>>> abs(3+4j)
5.0
```

chr(i) 함수

```
>>> chr(65)
'A'
>>> ord('A')
65
```

compile(source, filename, mode) 함수

```
>>> with open('mymodule.py') as f:
    lines = f.read()

>>> code_obj = compile(lines, 'mymodule.py', 'exec')
>>> exec(code_obj)
...
```

complex(real, imag) 함수

```
>>> x = complex(4, 2)
>>> x
(4+2j)
```

dir 함수

- **dir**은 객체가 가지고 있는 변수나 함수를 보여 준다.

```
>>> dir([1, 2, 3])
['__add__', '__class__', '__contains__', '__delattr__', '__delitem__', '__dir__',
'__doc__', '__eq__', '__format__', '__ge__', '__getattribute__', '__getitem__',
'__gt__', '__hash__', '__iadd__', '__imul__', '__init__', '__iter__', '__le__',
'__len__', '__lt__', '__mul__', '__ne__', '__new__', '__reduce__',
'__reduce_ex__', '__repr__', '__reversed__', '__rmul__', '__setattr__',
'__setitem__', '__sizeof__', '__str__', '__subclasshook__', 'append', 'clear',
'copy', 'count', 'extend', 'index', 'insert', 'pop', 'remove', 'reverse', 'sort']
```

eval() 함수

- **eval()**은 파이썬의 수식을 문자열로 받아서 실행하고 그 결과를 반환한다.

```
>>> eval("1+2");
3

>>> x = 1
>>> y = 1
>>> eval('x+y')
2

>>> eval("print('Hi!')")
Hi!
```

exec() 함수

- **exec()** 함수는 수식뿐만 아니라 모든 파이썬 문장을 받아서 실행한다.

```
>>> exec("y=2+3")
>>> y
5

>>> statements = '''
import math
def area_of_circle(radius):
        return math.pi * radius * radius

'''
>>> exec(statements)
>>> area_of_circle(5)
78.53981633974483
```

float() 함수

- **float()** 함수는 문자열을 부동소수점수로 변환하는 기능을 한다.

```
>>> str = input("실수를 입력하시오: ")
실수를 입력하시오: 12.345
>>> str
'12.345'
>>> value=float(str)
>>> value
12.345
```

max() 함수

- **max()** 함수는 리스트나 튜플, 문자열에서 가장 큰 항목을 반환한다.

```
>>> values = [ 1, 2, 3, 4, 5]
>>> max(values)
5
>>> min(values)
1
```

max() 함수

- **max()** 함수는 리스트나 튜플, 문자열에서 가장 큰 항목을 반환한다.

```
>>> values = [ 1, 2, 3, 4, 5]
>>> max(values)
5
>>> min(values)
1
```

파이썬에서 정렬하기

```
>>> sorted([4, 2, 3, 5, 1])
[1, 2, 3, 4, 5]

>>> myList = [4, 2, 3, 5, 1]
>>> myList.sort()
>>> myList
[1, 2, 3, 4, 5]
```

key 매개변수

- 정렬을 하다보면 정렬에 사용되는 키를 개발자가 변경해 주어야 하는 경우가 종종 있다.

```
>>> sorted("The health know not of their health, but only the sick".split(), key=str.lower)
['but', 'health', 'health,', 'know', 'not', 'of', 'only', 'sick', 'The', 'the', 'their']
```

```
>>> students = [
        ('홍길동', 3.9, 20160303),
        ('김철수', 3.0, 20160302),
        ('최자영', 4.3, 20160301),
]
>>> sorted(students, key=lambda student: student[2])
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```

예제

```
>>> class Student:
        def __init__(self, name, grade, number):
                self.name = name
                self.grade = grade
                self.number = number
        def __repr__(self):
                return repr((self.name, self.grade, self.number))

>>> students = [
        Student('홍길동', 3.9, 20160303),
        Student('김철수', 3.0, 20160302),
        Student('최자영', 4.3, 20160301),
]
>>> sorted(students, key=lambda student: student.number)
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9,
20160303)]
```

오름차순 정렬과 내림차순 정렬

```
>>> sorted(students, key=lambda student: student.number, reverse=True)
[('홍길동', 3.9, 20160303), ('김철수', 3.0, 20160302), ('최자영', 4.3, 20160301)]
```

Lab: 키를 이용한 정렬 예제

- 주소록을 작성한다고 하자. 간단하게 사람들의 이름과 나이만 저장하고자 한다. 사람을 **Person** 이라는 클래스로 나타낸다. **Person** 클래스는 다음과 같은 인스턴스 변수를 가진다.
  - name – 이름을 나타낸다.(문자열)
  - age – 나이를 나타내낸다.(정수형)
- 나이순으로 정렬하여 보여주는 프로그램을 작성하자. 정렬의 기준의 사람의 나이이다.

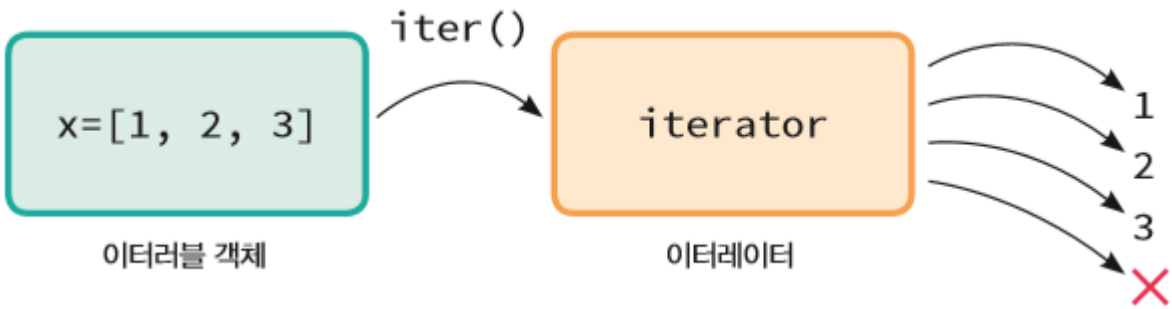
[<이름: 홍길동, 나이: 20>, <이름: 김철수, 나이: 35>, <이름: 최자영, 나이: 38>]

Solution

```
class Person(object):
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def __repr__(self):
        return "<이름: %s, 나이: %s>" % (self.name, self.age)

def keyAge(person):
    return person.age

people = [Person("홍길동", 20), Person("김철수", 35), Person("최자영", 38)]
print(sorted(people, key = keyAge))
```

이터레이터

- 파이썬에서는 **for** 루프와 함께 사용할 수 있는 여러 종류의 객체가 있으며 이들 객체는 이터러블 객체 **(iterable object)**이라고 불린다.



객체가 이터러블 객체가 되려면

- **__iter__ ()**은 이터러블 객체 자신을 반환한다.
- **__next__ ()**은 다음 반복을 위한 값을 반환한다. 만약 더 이상의 값이 없으면 **StopIteration** 예외를 발생하면 된다.

예제

```
class MyCounter(object):
    # 생성자 메소드를 정의한다.
    def __init__(self, low, high):
        self.current = low
        self.high = high

    # 이터레이터 객체로서 자신을 반환한다.
    def __iter__(self):
        return self

    def __next__(self):
        # current가 high 보다 크면 StopIteration 예외를 발생한다.
        # current가 high 보다 작으면 다음 값을 반환한다.
        if self.current > self.high:
            raise StopIteration
        else:
            self.current += 1
            return self.current - 1
```

예제

```
c = MyCounter(1, 10)
for i in c:
        print(i, end=' ')
```

1 2 3 4 5 6 7 8 9 10

제너레이터

- 제너레이터**(generators)**는 키워드 **yield**를 사용하여서 함수로부터 이터레이터를 생성하는 하나의 방법이다.

```
def myGenerator():
    yield 'first'
    yield 'second'
    yield 'third'
```

예제

```
def myGenerator():
    yield 'first'
    yield 'second'
    yield 'third'

for word in myGenerator():
    print(word)
```

```
first
second
third
```

클로저

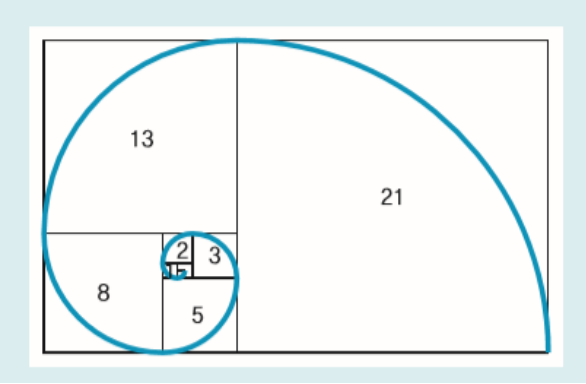
- 클로저(**closure**)는 함수에 의하여 반환되는 함수이다.

```
def addNumber(fixedNum):
  def add(number):
      return fixedNum + number
  return add

func = addNumber(10)
print(func(20))
```

```
30
```

Lab: 피보나치 이터레이터

- 피보나치 수열이란 앞의 두 수의 합이 바로 뒤의 수가 되는 수열을 의미한다. 피보나치 수열의 수들을 생성하는 이터레이터 클래스를 정의해보자.



1 1 2 3 5 8 13 21 34

Solution

```
class FibIterator:
    def __init__(self, a=1, b=0, maxValue=50):

        self.a = a
        self.b = b
        self.maxValue = maxValue
    def __iter__(self):
        return self

    def __next__(self):
        n = self.a + self.b
        if n > self.maxValue:
            raise StopIteration()
        self.a = self.b
        self.b = n
        return n

for i in FibIterator():
    print(i, end=" ")
```

연산자 오버로딩

- 연산자를 메소드로 정의하는 것을 연산자 오버로딩 **(operator overloading)**이라고 한다.

오버로딩할 수 있는 연산자

연산자	수식예	내부적인 함수 호출
덧셈	x + y	x.__add__(y)
뺄셈	x - y	x.__sub__(y)
곱셈	x * y	x.__mul__(y)
지수	x ** y	x.__pow__(y)
나눗셈(실수)	x / y	x.__truediv__(y)
나눗셈(정수)	x // y	x.__floordiv__(y)
나머지	x % y	x.__mod__(y)
비트 왼쪽 이동	x << y	x.__lshift__(y)
비트 오른쪽 이동	x >> y	x.__rshift__(y)
비트 AND	x & y	x.__and__(y)
비트 OR	x | y	x.__or__(y)
비트 XOR	x ^ y	x.__xor__(y)
비트 NOT	~x	x.__invert__()

예제

```
>>> s1="Impossible "
>>> s2="Dream"
>>> s3 = s1.__add__(s2)
>>> s3
'Impossible Dream'
>>>
```

```
>>> class Point:
    def __init__(self,x = 0,y = 0):
        self.x = x
        self.y = y

>>> p1 = Point(1, 2)
>>> p2 = Point(3, 4)
>>> p1 + p2
...
TypeError: unsupported operand type(s) for +: 'Point' and 'Point'
```

예제

```
>>> class Point:
    def __init__(self,x = 0,y = 0):
        self.x = x
        self.y = y

    def __add__(self, other):
        x = self.x + other.x
        y = self.y + other.y
        return Point(x, y)

    def __str__(self):
        return 'Point('+str(self.x)+', '+str(self.y)+')'

>>> p1 = Point(1, 2)
>>> p2 = Point(3, 4)
>>> print(p1+p2)
(4,6)
```

Lab: 피보나치 이터레이터

- 책을 나타내는 **Book** 클래스를 작성하고 **Book** 클래스 내부에 **__add__()** 함수를 정의하여서 **Book** 클래스의 객체들을 서로 합할 수 있게 하라. **__add__()** 함수는 책의 페이지들을 합쳐서 반환한다.

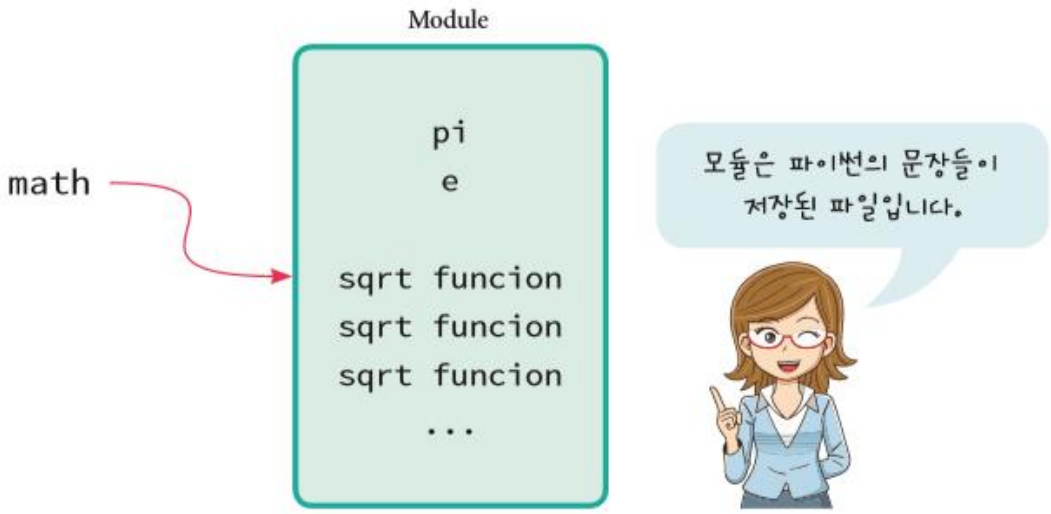
```
>>> book1 = Book('자료구조', 600)
>>> book2 = Book('C언어', 700)
>>> book1+book2
1300
```

Solution

```
class Book:
        title = ''
        pages = 0

        def __init__(self, title='', pages=0):
                self.title = title
                self.pages = pages

        def __str__(self):
                return self.title

        def __add__(self, other):
                return self.pages + other
```

모듈

- 파이썬에서 모듈(**module**)이란 함수나 변수 또는 클래스들을 모아 놓은 파일이다.



모듈 작성하기

fibo.py

```
# 피보나치 수열 모듈

def fib(n):           # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

def fib2(n):          # 피보나치 수열을 리스트로 반환
    result = []
    a, b = 0, 1
    while b < n:
        result.append(b)
        a, b = b, a+b
    return result
```

모듈 사용하기

```
>>> import fibo

>>> fibo.fib(1000)
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987

>>> fibo.fib2(100)
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]

>>> fibo.__name__
'fibo'
```

모듈 실행하기

```
C> python fibo.py <arguments>
```

```
...
if __name__ == "__main__":
    import sys
    fib(int(sys.argv[1]))
```

```
C> python fibo.py 50
1 1 2 3 5 8 13 21 34
```

모듈 탐색 경로

① 입력 스크립트가 있는 디렉토리(파일이 지정되지 않으면 현재 디렉토리)

② **PYTHONPATH** 환경 변수

③ 설치에 의존하는 디폴트값

유용한 모듈

- 프로그래밍에서 중요한 하나의 원칙은 이전에 개발된 코드를 적극적으로 재활용하자는 것

copy 모듈

```
import copy
colors = ["red", "blue", "green"]
clone = copy.deepcopy(colors)

clone[0] = "white"
print(colors)
print(clone)
```

```
['red', 'blue', 'green']
['white', 'blue', 'green']
```

keyword 모듈

```
import keyword

name = input("변수 이름을 입력하시오: ")

if keyword.iskeyword(name):
    print (name, "은 예약어임.")
    print ("아래는 키워드의 전체 리스트임: ")
    print (keyword.kwlist)
else:
    print (name, "은 사용할 수 있는 변수이름임.")
```

```
변수 이름을 입력하시오: for
for 은 예약어임.
아래는 키워드의 전체 리스트임:
['False', 'None', 'True', 'and', 'as', 'assert', 'break', 'class', 'continue', 'def', 'del',
'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda',
'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

random 모듈

```
>>> import random
>>> print(random.randint(1, 6))
6
>>> print(random.randint(1, 6))
3

>>> import random
>>> print(random.random()*100)
81.1618515880431

>>> myList = [ "red", "green", "blue" ]
>>> random.choice(myList)
'blue'
```

random 모듈

```
>>> myList = [ [x] for x in range(10) ]
>>> random.shuffle(myList)
>>> myList
[[3], [2], [7], [9], [8], [1], [4], [6], [0], [5]]

>>> for i in range(3):
        print(random.randrange(0, 101, 3))

81
21
57
```

os 모듈

```
>>> import os
>>> os.system("calc")
```

```
>>> os.getcwd()
'D:\\'
>>> os.chdir("D:\\tmp")
>>> os.getcwd()
'D:\\tmp’

>>> os.listdir(".")
[ 'chap01', 'chap02', 'chap03', 'chap04', 'chap05', 'chap06', 'chap07',
'chap08', 'chap09', 'chap10', 'chap11', 'chap12', 'chap13', 'chap14', 'chap15',
'chap16', 'chap17', 'chap18', 'chap19', 'chap20' ]
```

sys 모듈

```
>>> import sys
>>> sys.prefix # 파이썬이 설치된 경로
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-32’

>>> sys.executable
'C:\\Users\\chun\\AppData\\Local\\Programs\\Python\\Python35-
32\\pythonw.exe’
```

time 모듈

```
>>> import time
>>> time.time()
1461203464.6591916
```

예제

```
import time
def fib(n):    # 피보나치 수열 출력
    a, b = 0, 1
    while b < n:
        print(b, end=' ')
        a, b = b, a+b
    print()

start = time.time()
fib(1000)
end = time.time()
print(end-start)
```

1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
0.03500199317932129

calendar 모듈

```
import calendar

cal = calendar.month(2016, 8)
print(cal)
```

```
    August 2016
Mo Tu We Th Fr Sa Su
 1  2  3  4  5  6  7
 8  9 10 11 12 13 14
15 16 17 18 19 20 21
22 23 24 25 26 27 28
29 30 31
```

Lab: 동전 던기지 게임

- 하나의 예제로 동전 던지기 게임을 파이썬으로 작성해보자. **random** 모듈을 사용한다.

```
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
tail
동전 던지기를 계속하시겠습니까?( yes, no) yes
head
동전 던지기를 계속하시겠습니까?( yes, no) no
```

Solution

```
import random
myList = [ "head", "tail" ]

while (True):
        response = input("동전 던지기를 계속하시겠습니까?( yes, no) ");
        if response == "yes":
                coin = random.choice(myList)
                print (coin)
        else :
                break
```

헥심 정리

- 파이썬에는 어떤 객체에도 적용이 가능한 내장 함수가 있다. **len()**나 **max()**와 같은 함수들을 잘 사용하면 프로그래밍이 쉬워진다.
- 클래스를 정의할 때 **__iter__(self)**와 **__next__(self)** 메소드만 정의하면 이터레이터가 된다. 이터레이터는 **for** 루프에서 사용할 수 있다.
- 연산자 오버로딩은 **+**나 **–**와 같은 연산자들을 클래스에 맞추어서 다시 정의하는 것이다. 연산자에 해당되는 메소드(예를 들어서 **__add__(self, other)**)를 클래스 안에서 정의하면 된다.

Q & A