

# 11 장 내장함수, 람다식, 제너레이터, 모듈

# 학습 목표

- 파이썬에 내장된 내장 함수들을 살펴본다.
- 람다식을 살펴본다.
- 제너레이터를 사용하여 반복 가능한 객체를 작성할 수 있다.
- 모듈의 개념을 살펴본다.
- 유용한 모듈을 사용할 수 있다.



# 내장 함수

Python » Korean » 3.9.6 » Documentation »

이전 항목

10. 전체 문법 규칙

다음 항목

소개

현재 문서

버그 보고하기  
소스 보기

## 파이썬 표준 라이브러리

파이썬 언어 레퍼런스는 파이썬 언어의 정확한 문법과 의미를 설명하고 있지만, 이 라이브러리 레퍼런스 설명서는 파이썬과 함께 배포되는 표준 라이브러리를 설명합니다. 또한, 파이썬 배포판에 일반적으로 포함되어있는 선택적 구성 요소 중 일부를 설명합니다.

파이썬의 표준 라이브러리는 매우 광범위하며, 이 라이브러리에는 일상적인 프로그래밍에서 발생하는 성된 모듈뿐만 아니라, 파일 I/O와 같은 시스템 기술들이 없다면 파이썬 프로그래머가 액세스할 방법 없는 API들로 추상화시킴으로써, 파이썬 프로그램이 가능하게 합니다.

윈도우 플랫폼용 파이썬 설치 프로그램은 일반적으로도 포함합니다. 유닉스와 같은 운영체제의 경우, 표준 라이브러리와 함께 제공되는 패키지 도구를 사용하여 선택적으로도 포함합니다.

표준 라이브러리 외에도, 수천 가지 컴포넌트(개별 임포트)가 늘어나고 있는데, 파이썬 패키지 색인

- 소개
  - 가용성에 대한 참고 사항
- 내장 함수
- 내장 상수
  - site 모듈에 의해 추가된 상수들
- 내장형
  - 논리값 검사
  - 논리 연산 — and, or, not
  - 비교
  - 숫자 형 — int, float, complex
  - 이터레이터 형
  - 시퀀스 형 — list, tuple, range
  - 텍스트 시퀀스 형 — str
  - 바이너리 시퀀스 형 — bytes, bytearray

## 내장 함수

파이썬 인터프리터에는 항상 사용할 수 있는 많은 함수와 형이 내장되어 있습니다. 여기에서 알파벳 순으로 나열합니다.

내장 함수				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	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()	

# 내장함수

- |          |       |                     |          |
|----------|-------|---------------------|----------|
| □ abs()  | all() | any()               | bin()    |
| □ eval() | sum() | len()               | list()   |
| □ map()  | dir() | complex(real, imag) |          |
| □ max()  | min() | enumerate()         | filter() |
| □ zip()  |       |                     |          |

p505.py

# Lab: 내장 함수 예제

- 파티이 초대장을 2개의 리스트로 저장한다. 첫번째 리스트에는 참석자의 이름, 두번째 리스트에는 동반자들의 숫자가 저장되어 있다.

```
>>> invitations = ["Kim", "Lee", "Park", "Choi"]
```

p511.py

```
>>> persons = [1, 3, 0, 6]
```

```
>>> sum(persons)
```

```
10
```

파티에 한 사람이라도 오는 지를 확인해보자. 이것은 any() 함수로 가능하다.

```
>>> any(persons)
```

```
True
```

이번에는 모든 초대받은 그룹이 전부 오는 지를 확인해보자. 이것은 all() 함수로 가능하다.

```
>>> all(persons)
```

```
False
```

```
...
```

```
...
```

# 정렬과 탐색

- `sorted()` : 기존의 리스트를 변경하는 것이 아니라 정렬된 새로운 리스트를 반환. 리스트 외의 반복가능 객체(딕셔너리)에서도 사용 가능
- `sort()` : 리스트 자체를 변경. 리스트에서만 사용 가능

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

p512.py

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

# key 매개변수

p512.py

- 정렬에 사용되는 키를 변경해주어야 하는 경우에 사용
- 문자열을 받아서 `split()`로 단어들의 리스트로 변환한 후 `key`를 문자열 객체의 `lower()` 함수로 지정

```
>>> 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']
```

- 학생들의 학번을 기준으로 정렬

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

```
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```

# 오름차순 정렬과 내림차순 정렬

p512.py

- reverse 변수가 True이면 내림차순 정렬

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

```
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```



# 추가 예제 (책에 없음)

```
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),
]
print(sorted(students, key=lambda student: student.number) )
```

p514.py

```
[('최자영', 4.3, 20160301), ('김철수', 3.0, 20160302), ('홍길동', 3.9, 20160303)]
```

# Lab: 키를 이용한 정렬 예제

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

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

```
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)
```

p515.py

...

...

# 람다식

- 이름은 없고 몸체만 있는 함수. **lambda** 키워드로 만들어진다. 딱 한번 사용하는 함수를 만드는데 사용된다.

Syntax: 람다식 정의

**형식** lambda 매개 변수들: 수식

**예** lambda x, y: x+y;

매개 변수

함수의 몸체

p516.py

```
f = lambda x, y: x+y;
```

```
print( "정수의 합 : ", f( 10, 20 ))
```

```
print( "정수의 합 : ", f( 20, 20 ))
```

=

```
def get_sum(x, y):  
    return x+y
```

```
print( "정수의 합 : ", get_sum( 10, 20 ))
```

```
print( "정수의 합 : ", get_sum( 20, 20 ))
```

```
정수의 합 : 30
```

```
정수의 합 : 40
```

# 람다식과 콜백 함수

- 이벤트가 생성되면 호출되는 함수를 전달할 때 람다식을 사용

```
from tkinter import *
```

p517\_lambda.py

```
window = Tk()
```

```
btn1 = Button(window, text="1 출력", command=lambda: print(1, "버튼이 클릭"))
```

```
btn1.pack(side=LEFT)
```

```
btn2 = Button(window, text="2 출력", command=lambda: print(2, "버튼이 클릭"))
```

```
btn2.pack(side=LEFT)
```

```
quitBtn = Button(window, text="QUIT", fg="red", command=quit)
```

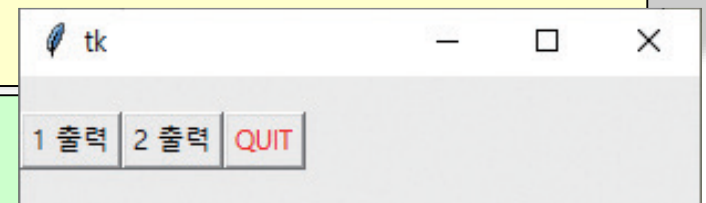
```
quitBtn.pack(side=LEFT)
```

```
mainloop()
```

```
2 버튼이 클릭
```

```
1 버튼이 클릭
```

```
2 버튼이 클릭
```



- 람다식은 내장 함수와 함께 사용된다.
- `map()` 함수와 람다식

```
list_a = [ 1, 2, 3, 4, 5 ]  
f = lambda x : 2*x  
result = map(f, list_a)  
print(list(result))
```

[2, 4, 6, 8, 10]

- `filter()` 함수와 람다식

```
list_a = [1, 2, 3, 4, 5, 6]  
result = filter(lambda x : x % 2 == 0, list_a)  
print(list(result))
```

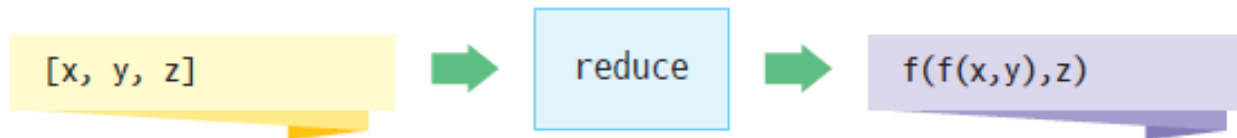
[2, 4, 6]

```
data = [(3, 100), (1, 200), (7, 300), (6, 400)]  
sorted(data, key=lambda item: item[0])  
print(data)
```

[(3, 100), (1, 200), (7, 300), (6, 400)]

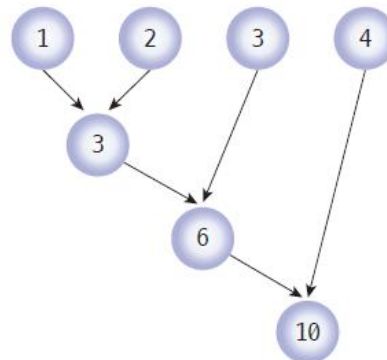
# reduce() 함수와 람다식

- `reduce(func, seq)` 함수는 `func()` 함수를 시퀀스 `seq`에 연속적으로 적용하여 단일 값을 반환한다.



```
import functools
result = functools.reduce(lambda x,y: x+y, [1, 2, 3, 4])
print(result)
```

10



# Lab: 람다식으로 온도 변환하기

p519.py

- 람다식과 `map()` 함수를 사용하여서 화씨 온도를 섭씨 온도로 변환하는 명령문을 작성해보자.
- 람다식을 사용하지 않는 코드

```
def celsius(T):  
    return (5.0/9.0)*(T-32.0)  
  
f_temp = [0, 10, 20, 30, 40, 50]  
c_temp = map(celsius, f_temp)  
print(list(c_temp))
```

- 람다식을 사용하는 코드

```
f_temp = [0, 10, 20, 30, 40, 50]  
c_temp = map(lambda x: (5.0/9.0)*(x-32.0), f_temp)  
print(list(c_temp))
```

# Lab: 람다식으로 데이터 처리하기

- 옷가게. 아래의 데이터를 처리하여서 2개의 튜플이 저장된 리스트를 반환하는 프로그램을 작성하라

주문 번호	상품명	수량	상품 당 가격
1	"재킷"	5	120000
2	"셔츠"	6	24000
3	"바지"	3	50000
4	"코트"	6	300000

```
[('1', 600000), ('2', 144000), ('3', 150000), ('4', 1800000)]
```

```
orders = [ ["1", "재킷", 5, 120000],  
            ["2", "셔츠", 6, 24000],  
            ["3", "바지", 3, 50000],  
            ["4", "코트", 6, 300000] ]
```

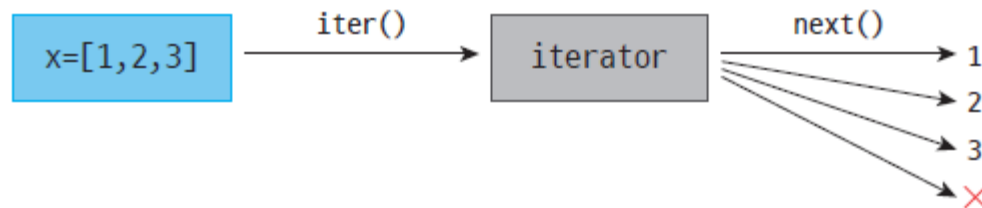
p520.py

```
result = list(map(lambda x: (x[0], x[2] * x[3]), orders))  
print(result)
```



# 이터레이터(iterator)

- 이터러블 객체(iterable object) : for 루프와 함께 반복 가능한 객체



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

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

```
...  
...
```

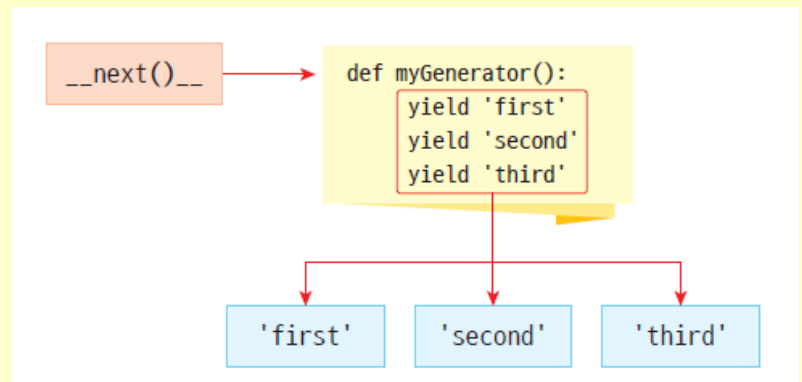
p521.py

# 제너레이터(generator)

- 함수로부터 반복가능한 객체, 이터레이터를 생성하는 것
- 키워드 `yield` 사용

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

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



```
def MyCounterGen(low, high):  
    while low <= high:  
        yield low  
        low += 1
```

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

p523.py

# 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

```
class Fiblterator:
    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
        ...
        ...
```

p524.py

# 연산자 오버로딩

- 연산자 오버로딩(operator overloading) : 연산자를 메소드로 정의하는 것

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

연산자	수식에	내부적인 함수 호출
덧셈	$x + y$	<code>x.__add__(y)</code>
뺄셈	$x - y$	<code>x.__sub__(y)</code>
곱셈	$x * y$	<code>x.__mul__(y)</code>
지수	$x ** y$	<code>x.__pow__(y)</code>
나눗셈(실수)	$x / y$	<code>x.__truediv__(y)</code>
나눗셈(정수)	$x // y$	<code>x.__floordiv__(y)</code>
나머지	$x \% y$	<code>x.__mod__(y)</code>
작음	$x < y$	<code>x.__lt__(y)</code>
작거나 같음	$x \leq y$	<code>x.__le__(y)</code>
같음	$x == y$	<code>x.__eq__(y)</code>
같지 않음	$x != y$	<code>x.__ne__(y)</code>
큼	$x > y$	<code>x.__gt__(y)</code>
크거나 같음	$x \geq y$	<code>x.__ge__(y)</code>

# 클래스에서 연산자 정의하기

- 점과 점을 + 연산으로 합할 수 있도록 연산자 오버로딩

```
class Point:
```

p526.py

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

```
# + 연산자와 print 연산자를 재정의(연산자 오버로딩)하여 점과 점의 연산을 가능하게 구현
```

```
# 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)
```

Point(4, 6)

# Lab: Book 클래스

- 책의 페이지 수를 기준으로 2개의 **Book** 객체를 비교
- > 연산자와 **print()**를 재정의(연산자 오버로딩)

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

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

    def __str__(self):
        return self.title

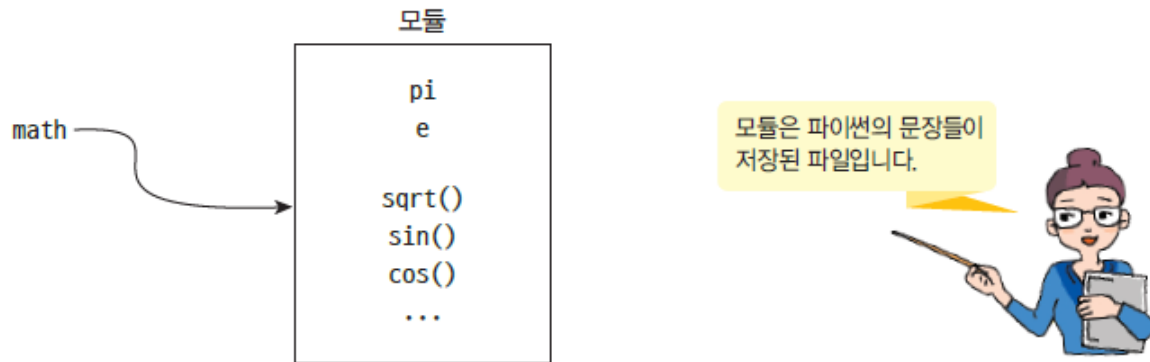
    def __gt__(self, other):
        return self.pages > other.pages
```

p527.py

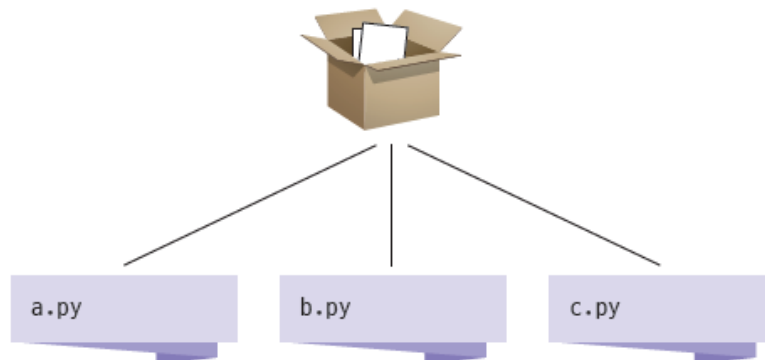
...  
...

Master of Python

- 모듈(module) : 함수나 변수 또는 클래스 들을 모아 놓은 파일



- 파이썬 프로그램이 길어지면, 유지 보수를 쉽게 하기 위해 여러 개의 파일로 분할 할 수 있다. 또한 파일을 사용하면 한번 작성한 함수를 복사하지 않고 여러 프로그램에서 사용할 수 있다.



# 모듈 작성하기

모듈 작성

모듈 사용

*fibonacci.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
```

```
# 1
import fibo

fibo.fib(1000)
print(fibo.fib2(100))

# 2
from fibo import fib
fib(1000)

#3
from fibo import *
fib(500)
```

```
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
[1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89]
1 1 2 3 5 8 13 21 34 55 89 144 233 377 610 987
1 1 2 3 5 8 13 21 34 55 89 144 233 377
```



# 모듈의 별칭, 실행하기

- 모듈의 별칭을 만들어서 사용할 수도 있다.

```
import mymodule as lib
...
lib.func(100)
```

- 모듈 실행하기 - 모듈의 끝에 아래 코드가 추가되면 모듈을 명령 프롬프트에서 실행할 수 있다.

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

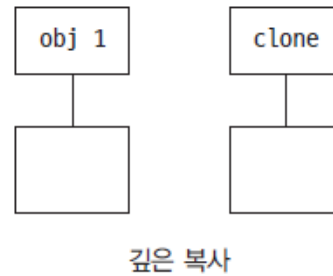
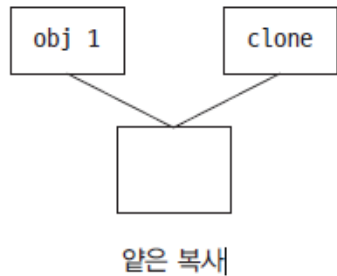
```
C> python fibo.py 50
1 1 2 3 5 8 13 21 34
```

# 유용한 모듈

- 잘 정리되고 충분한 테스트를 거친 좋은 모듈이 제공되고 있는데 굳이 코드를 재작성할 필요는 없다.
- 프로그래밍에서 중요한 하나의 원칙은 이전에 개발된 코드를 적극적으로 재활용하는 것이다.

# copy 모듈

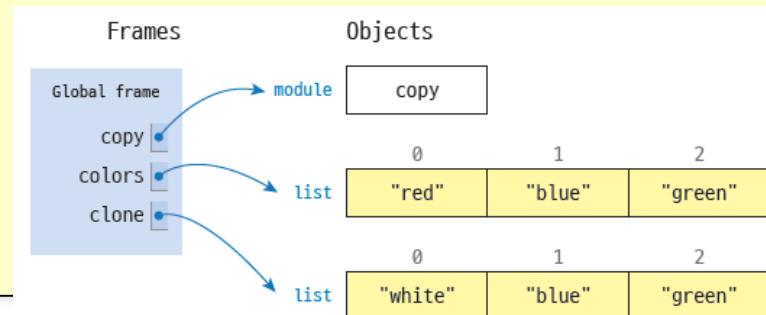
- 얇은 복사(shallow copy) - 객체의 참조값만 복사되고 객체 자체는 복사되지 않는다.
- 깊은 복사(deep copy) - 객체까지 복사된다. `deepcopy(object)` 사용



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

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

p534\_deepcopy.py



# random 모듈

- ❑ randint() : 정수 범위의 난수를 생성
- ❑ random() : 0에서 1 사이의 난수를 생성
- ❑ choice() : 주어진 시퀀스 항목을 랜덤하게 선택
- ❑ shuffle() : 리스트의 항목을 랜덤하게 섞는다.
- ❑ randrange(start, stop[, step]) : range(start, stop, step) 구간으로 부터 랜덤하게 요소를 생성

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

print(random.random() * 100)

myList = ["red", "green", "blue"]
print(random.choice(myList))

...
```

p534\_deepcopy.py

# sys 모듈

- 파이썬 인터프리터에 대한 다양한 정보를 제공하는 모듈

```
import sys p535.py  
  
print(sys.prefix)    #파이썬이 설치된 경로  
  
print("---")  
print(sys.executable) #실행파일의 이름  
  
print("---")  
print(sys.path)       #모듈을 참조할 때 사용하는 경로  
  
print("---")  
print(sys.version)    #설치된 파이썬의 버전
```

```
C:\ProgramData\Anaconda3  
---  
C:\ProgramData\Anaconda3\python.exe  
---  
['C:\\ProgramData\\Anaconda3\\python38.zip', 'C:\\ProgramData\\Anaconda3\\DLLs',  
'C:\\ProgramData\\Anaconda3\\lib',  
'C:\\ProgramData\\Anaconda3', '',  
'C:\\ProgramData\\Anaconda3\\lib\\site-packages', ...  
---  
3.8.5 (default, Sep  3 2020, 21:29:08) [MSC  
v.1916 64 bit (AMD64)]
```

# time 모듈

- 시간을 다양한 형식으로 표시하는 함수들을 가진 모듈

```
# 1
import time
print(time.time()) #1970.1.1 이후부터 지금까지 흘러온 시간(초)
```

p536.py

```
# 2
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) #실행시간을 측정
```

# time 모듈

#3

```
import time
```

```
print(time.asctime()) #현재 날짜와 시간을 문자열 형태로 출력
```

# 4

```
import time
```

```
t = (2016, 4, 29, 12, 10, 50, 5, 0, 0) #(연, 월, 일, 시, 분, 초, 요일, 0, 0) 형식으로 날짜 지정
```

```
print(time.asctime(t))
```

# 5

```
import time
```

```
for i in range(10, 0, -1):
```

```
    print(i, end=" ")
```

```
    time.sleep(1) #1초 동안 정지
```

```
print("발사! ")
```

# calendar

```
import calendar
```

p538.py

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

```
print(cal)
```

```
cal = calendar.calendar(2021)
```

```
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
```

2021

```
January
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
```

```
February
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
```

```
March
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
```

```
April
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
```

```
May
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
```

```
June
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
```



# keyword 모듈(교재에 없음)

- 파이썬 예약어를 확인하는 모듈

```
import keyword
```

p539\_aa.py

```
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']
```

# Lab: 동전 던지기 게임

```
import random
myList = [ "앞면", "뒷면" ]
```

p539\_coin\_toss.py

```
while (True):
    response = input("동전 던지기를 계속하시겠습니까?(y, n) ");
    if response == "y":
        coin = random.choice(myList)
        print(coin)
    else :
        break
```

```
동전 던지기를 계속하시겠습니까?(y, n) y
앞면
```

```
동전 던지기를 계속하시겠습니까?(y, n) y
뒷면
```

```
...
```

# 이번 장에서 배운 것

- 파이썬에는 어떤 객체에도 적용이 가능한 내장 함수가 있다. `len()`나 `max()`와 같은 함수들을 잘 사용하면 프로그래밍이 쉬워진다.
- 클래스를 정의할 때 `(self)`와 `(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.

