Ch5-1. 클래스 (Class)

- **클래스**(class)는 객체를 표현하기 위한 문법으로, 똑같은 무엇인가를 계속해서 만들어 낼수 있는 설계 도면이다.
- **객체**(object)란 클래스(class)로 만든 결과물이다.

- 쿠키를 만드는 쿠키 틀과 그것을 사용해 만든 쿠키를 클래스와 객체로 비유 하여 볼 수 있다.
  - 쿠키 틀 → 클래스 (class)
  - 쿠키 틀에 의해서 만들어진 쿠키 → 객체 (object)

- 클래스로 만든 객체의 특징
  - 객체마다 고유한 성격을 가짐
  - 쿠키 틀로 만든 쿠키를 조금 잘라 먹어도 다른 쿠키에는 아무 영향이 없는 것과 마찬가지로 동일한 클래스로 만든 객체들은 서로 전혀 영향을 주지 않는다.

Table of Contents

1. Class는 왜 필요한가?

- 클래스(class) 란 객체 지향(object oriented) 프로그래밍을 구현함에 있어서 필수적인 개념이다.
- 여러 변수와 함수들을 하나로 구조화 시켜 동일한 작업을 계속해서 해야할 때 강력한 힘을 발휘한다.

계산기의 "더하기" 기능을 구현한 파이썬 코드로 두 대의 계산기를 만들어 보고자 한다. 이전에 계산한 결괏값을 유지하기 위해서 result 전역 변수(global)를 사용하여 아래와 같이 만들었다.

In [70]:
```
result1 = 0

def add1(num):
  global result1
  result1 += num
  return result1
```

In [71]:
```
print(add1(3))
```

```
3
```

In [72]:
```
print(add1(5))
```

```
8
```

In [73]:
```
#두 개의 계산기 예제
result1 = 0
result2 = 0

def add1(num):
  global result1
  result1 += num
  return result1

def add2(num):
  global result2
  result2 += num
  return result2
```

In [74]:
```
print(add1(3))
print(add1(5))
```

```
3
8
```

In [75]:
```
print(add2(3))
print(add2(7))
```

```
3
10
```

각 계산기는 각각 하나의 결괏값을 유지해야 하기 때문에 위와 같이 add 함수 하나만으로는 결괏값을 따로 유지할 수 없다. 따라서, 두 개의 함수를 각각 따로 add1 과 add2 로 만들었고, 각 함수에서 계산한 결과값을 유지하면서 저장하는 전역변수 result1, restul2가 필요하다.

하지만, 위와 같은 경우에 클래스를 사용하면 다음과 같이 간단하게 해결할 수 있다.

In [78]:
```
# class를 사용하는 경우 2개의 계산값을 반환해야 할 때
class Calculator:
  def __init__(self):
    self.result = 0
  def add(self, num):
    self.result += num
    return self.result

cal1 = Calculator()
print(cal1.add(3))
print(cal1.add(4))
```

```
3
7
```

In [80]:
```
cal2 = Calculator()
print(cal2.add(3))
print(cal2.add(7))
```

```
3
10
```

클래스 Calculator로 만든 별개의 계산기 cal1, cal2 (파이썬에서는 이것을 객체라고 부른다)가 각각의 역할을 수행한다. 그리고 계산기(cal1, cal2)의 결괏값 역시 다른 계산기의 결괏값과 상관없이 독립적인 값을 유지한다. 클래스를 사용하면 계산기 대수가 늘어나더라도 객체를 생성만 하면 되기 때문에 함수를 사용하는 경우와 달리 매우 간단해진다.

2. 클래스(class) 기본구조

```
class 클래스명 :
  클래스 변수 1
  클래스 변수 2
    ...
  def 클래스 함수1(self, 입력요소1, 입력요소2, ...):
    ...
  def 클래스 함수2(self, 입력요소1, 입력요소2, ...):
    ...
```

In [ ]:
```
# Class를 사용하지 않을 때 : 동일한 작업을 계속해서 반복
car_brand1 = 'Hyundai'
```

```
car_color1 = 'white'
car_year1 = 1996
car_brand2 = 'Kia'
car_color2 = 'black'
car_year2 = 1998
car_brand3 = 'Toyota'
car_color3 = 'blue'
car_year3 = 2010
```

In [2]:
```
# Class 사용 : 구조화 시켜 객체 지향 프로그래밍을 가능하게 함
class car :
    message = "Car information"
    def set_info(self, brand, color, year) :
        self.brand = brand
        self.color = color
        self.year = year
```

In [3]:
```
car1 = car()
car2 = car()
car3 = car()

car1.set_info('Hyundai', 'white', 1996)
car2.set_info('Kia', 'black', 1998)
car3.set_info('Toyota', 'blue', 2010)
```

In [4]:
```
car.message
```

Out[4]: 'Car information'

In [5]:
```
car1.brand
```

Out[5]: 'Hyundai'

In [6]:
```
car2.color
```

Out[6]: 'black'

In [7]:
```
car3.year
```

Out[7]: 2010

3. 클래스(class) 만들기

예제를 통해 클래스를 어떻게 만드는지 알아보자.

클래스로 객체 만들기

Tips : 클래스로 만든 객체 (Object)를 **인스턴스** (Instance) 라고도 한다.
a = Cookie() 이렇게 만든 a는 객체이다. 그리고 a 객체는 Cookie의 인스턴스이다.
즉 인스턴스라는 말은 특정 객체(a)가 어떤 클래스(Cookie)의 객체인지를 관계 위주로 설명할 때 사용한다.

"a는 인스턴스"보다는 "a는 객체"라는 표현이 어울리며 "a는 Cookie의 인스턴스" 라는 표현이 훨씬 잘 어울린다.

In [8]:
```
class car :
  pass
```

In [9]:
```
a = car()
print(type(a))  # 객체 a의 타입은 car 클래스의 객체이나, 아직은 어떤 역활도 하지 못한다.
```

```
<class '__main__.car'>
```

객체에 숫자를 지정할 수 있게 만들기

- 메서드(Method)
  - 클래스 안에 구현된 함수, 클래스의 method는 도트 연산자(.)를 사용하여 호출한다.
- self
  - 클래스(class) 내부에 정의되는 함수의 첫 번째 입력요소는 반드시 self가 되어야 한다.
  - 즉, self는 함수를 부르는 객체가 해당 클래스의 인스턴스인지 확인해주는 장치이다.

In [11]:
```
class car :
  message = "Stop !!"
  def set_info(self, color, year):  # 메서드
    self.color = color
    self.year = year
```

앞에서 만든 car 클래스에서 pass 문장을 삭제하고 그 대신 set_info 함수를 만들었다.
클래스안에 구현된 함수는 다른 말로 메서드(Method)라고 부른다.

다음과 같이 a 객체를 만들고 a 객체를 통해 set_info 메서드를 호출해 보자.

In [12]:
```
a =car()   # a = car() 하는 순간 self 자리에 a가 들어가게 되고, a가 car클래스의 인스턴스임이 확인되어 클래스가 작동
a.set_info("Red", 2017)  # 객체를 통해 클래스의 메서드를 호출하려면 a.set_info("Red", 2017)와 같이 도트(.) 연산자를 사용
```

a.set_info 메서드는 매개변수로 self, color, year 3개 입력값을 받는데, 실제로는
a.set_info("Red", 2017) 처럼 2개 값만 전달 했다. 왜 그럴까? 첫 번째 매개변수 self에는
a.set_info 메서드를 호출한 객체 a가 자동으로 전달되기 때문이다.
일반 함수와는 달리 메서드의 첫 번째 매개변수 self는 특별한 의미를 가진다.

다음을 실행 해보자.

In [13]:
```
a = car()
b = car()
```

In [14]:
```
a.set_info("Red", 2017)
print(a.color)
```

Red

In [15]:
```
b.set_info("Blue", 2018)
print(b.color)
```

Blue

In [17]:
```
b.message
```

Out[17]: 'Stop !!'

기능 만들기

In [20]:
```
class car :
    message = "Stop !!"

    def set_info(self, color, year):
        self.color = color
        self.year = year

    def get_info(self):
        print("color is %s, year is %d" %(self.color, self.year))
```

In [21]:
```
new1 = car()
new2 = car()

new1.set_info("Red", 2017)
new2.set_info("Blue", 2018)

print(new1.color)
print(new2.year)
```

Red
2018

In [22]:
```
new1.get_info()
```

color is Red, year is 2017

In [23]:
```
new2.get_info()
```

color is Blue, year is 2018

4. 생성자 (Constructor)

위에서 만든 car 클래스를 new1.set_info와 같은 메서드(method)를 수행하지 않고, 직접 초기값을 설정하고 싶어서 아래와 같은 코드를 실행해 보았다.

In [25]:
```
a = car("Red", 2017)
```

```
---------------------------------------------------------------------------
TypeError                                 Traceback (most recent call last)
<ipython-input-25-e7206caa0a4a> in <module>
----> 1 a = car("Red", 2017)

TypeError: car() takes no arguments
```

위에서 보는 것과 같이 직접 입력값을 넣을 수가 없다.

클래스 자체엔 매개변수 입력값이 없고, set_info 메서드를 수행해야만 a의 객체변수 color 와 year가 생성할 수있게 만들어 두었기 때문이다.

따라서, Class에 직접적으로 초기값을 설정하기 위해서는 **생성자**(constructor)를 사용하는 것이 안전한 방법이다. 생성자란 객체가 생성될 때 자동으로 호출되는 매서드를 의미하며, 파이썬 매서드 이름으로 __init__ 를 사용하면 이 매서드는 생성자가 된다.

- __init__
  - 초기화 매서드 혹은 생성자라고 불리우는데, 인스턴스 생성 초기에 변수를 지정하는 것을 도와주는 장치이다.
  - 또한, 이러한 생성자는 인스턴스를 생성할 때 반드시 실행되는 구문으로, 보통 특정 입력 요소의 누락을 방지하기 위해 사용한다.

In [27]:
```
class car :
  message = "Stop !!"
  def __init__(self, color, year):   # 초기값 설정
    self.color = color
    self.year = year
  def get_info(self):
    print("color is %s, year is %d" %(self.color, self.year))
```

__init__ 메서드는 이전의 set_info 메서드와 이름만 다르고 모든 게 동일하다.
단 메서드 이름을 __init__ 으로 했기 때문에 생성자로 인식되어 객체가 생성되는 시점에 자동으로 호출되는차이가 있다.

In [28]:
```
new2 = car("Red", 2017)
```

In [29]:
```
new2.get_info()
```

```
color is Red, year is 2017
```

5. 클래스의 상속 (inheritance)

- 상속 (inheritance)이란 어떤 클래스를 만들 때 이미 만들어져 있는 다른 클래스의 기능을 물려받을 수있게 만드는 것이다.

- class 상속은 기존 class를 변경하지 않고 기능을 추가하거나 기존 기능을 변경하려고 할 때 사용한다.
- 기존 class를 직접 수정해도 되지만 기존 class가 라이브러리 형태로 제공되거나 수정이 허용되지 않는 상황이라면 상속을 사용한다.

- **클래스 상속 기본형**

  class 자식클래스 이름 (부모 클래스명) :

이미 만들어져 있는 car를 사용하여 (상속 개념을 사용하여) modify_car 클래스를 만들어 보자.

In [30]:
```
# modify_car 클래스는 car 클래스를 상속
class modify_car(car) :
  pass
```

In [31]:
```
# car 클래스의 모든 기능을 사용 할 수 있다.
a = modify_car("Red", 2017)
a.get_info()
```

```
color is Red, year is 2017
```

In [33]:
```
class modify_car(car):                    # car를 상속하여
  def get_color(self):                    # 새로운 메서드 추가
    return self.color
  def get_year(self):                     # 새로운 메서드 추가
    return self.year
```

In [34]:
```
a = modify_car("Red", 2017)
a.get_info()
```

```
color is Red, year is 2017
```

In [35]:
```
print(a.get_color())
print(a.get_year())
```

```
Red
2017
```

In [36]:
```
a.message
```

Out[36]:
```
'Stop !!'
```

상속은 modify_car 클래스처럼 기존 클래스(car)는 그대로 놔둔 채 클래스의 기능을 확장시킬 때 주로 사용한다.

6. 메서드 오버라이딩 (over-riding)

- 클래스 상속할 때 부모 class에 있는 메서드를 동일한 이름으로 자식 class에서 다시 만드는 것을 메서드 오버라이딩(Over-riding, 덮어쓰기)이라고 함
- 다시 말해, 부모 클래스(class)를 상속받아 변형하여 자식 클래스(class)를 생성하는 것을 말한다.

In [37]:
```
# 예제: get_info 를 year, color 순으로 print 하는 것으로 기존 class를 수정하여 상속하는 class를 생성

class modify_car_new(car):   # 클래스 상속
    message = 'Please stop !!'
    def get_info(self):          # 부모 클래스에 있던 매서드 변경 -> 오버라이딩
        print("year is %d, color is %s" % (self.year,self.color))
```

In [38]:
```
a = modify_car_new("Red", 2017)
a.get_info()
```

year is 2017, color is Red

In [39]:
```
a.message
```

Out[39]: 'Please stop !!'

7. 클래스 변수

- class 변수는 class 내에 함수(메서드)를 선언하는 것과 마찬가지로 class 내에 변수를 선언하여 생성한다.
- 이미 알고 있는 객체변수는 다른 객체들에 영향받지 않고 독립적으로 그 값을 유지하지만 클래스 변수는 객체 변수와 성격이 다르다.

1) 클래스 변수 생성

아래와 같이 lastname 이라는 클래스 변수를 만들 수 있고, "클래스이름.클래스 변수" 로 사용하거나 클래스로 만든 객체를 통해서도 클래스 변수를 사용할 수 있다.

In [48]:
```
class Family:
    lastname = "김"
print(Family.lastname)
```

김

In [49]:
```
a = Family()
b = Family()
print(a.lastname)
print(b.lastname)
```

김
김

2) 클래스 변수 변경

In [50]:
```
Family.lastname = "한"
```

In [51]:
```
print(a.lastname)
print(b.lastname)
```

```
한
한
```

중요한 것은 class 변수 값을 변경하면 class로 만든 객체의 lastname 값도 모두 변경된다는 것을 확인할 수 있다. 즉 class 변수는 class로 만든 모든 객체에 공유된다는 특징이 있다. (인스턴스 변수의 차이점)

위의 예제에서, 클래스 변수가 아닌 인스턴스 변수 입장에서 lastname을 변경한다면 아래와 같다.

In [52]:
```
a.lastname = "박"
```

이때, b.lastname 의 결과는 무엇인지 생각해보자.
b.lastname

즉, 인스턴스 변수에 저장된 데이터는 클래스의 모든 인스턴스에 공유되지 않는다.

Ch5-2. 모듈(Module)

- 모듈 (module)이란 함수나 변수 또는 클래스를 모아 놓은 python 파일이다.
- 다른 파이썬 프로그램에서 모듈 (module)을 불러와 사용할 수 있는 장점이 있다.
- 다양한 모듈 (module)을 설치하여 사용하면 손 쉬운 연산, 데이터 분석이 가능하다.

Table of Contents

1. 모듈의 생성

모듈에 대해 자세히 살펴보기 전에 간단한 모듈을 한번 만들어 보자.

In [1]:
```
# mod1.py
def add(a, b):
    return a+b
def sub(a, b):
    return a-b
```

- 위와 같이 add와 sub 함수만 있는 python 파일 mod1.py를 만들고, 이를 정해진 디렉토리에 저장하면 이 mod1.py 파일이 바로 모듈 (module) 이다.
- 즉, python 확장자 .py로 만든 python 파일은 모두 모듈 (module) 이며,
- 메모장에 위 코드를 적은 후 mod1.py로 저장해도 모듈 (module) 생성이 가능하다.

아래와 같이 class나 변수 등을 포함한 모듈 (module)도 만들 수 있다.

In [6]:
```
# mod2.py

PI = 3.141592

class Math:
    def solv(self, r):
        return PI*(r**2)

def add(a, b):
    return a+b
```

2. 모듈 불러오기

우리가 만든 mod1.py 파일, 즉 모듈을 파이썬에서 불러와 사용하려면 어떻게 해야할까?

1) cmd 창 (명령 프롬프트 창)을 이용하는 경우

명령 프롬프트 창(cmd 창)을 열고 mod1.py를 저장한 디렉터리로 이동 후 대화형 인터프리터를 실행한다.

cd C:\do
dir
python
import module이름

2) 쥬피터 노트북을 이용하는 경우

- 먼저 ipynb 파일이 있는 곳 (쥬피터 노브북이 저장된 경로) 으로 py 파일을 옮겨주고 ,
- "import module이름" 을 적어 모듈을 불러올 수 있다.
- 이때, module 이름은 확장자 명(.py)을 제외 하고 입력한다.

In [2]:
```
import mod1                    # module 이름은 확장자 명(.py)을 제외 하고 입력
```

module 내 함수를 사용하고자 하는 경우
- "module명.함수명" 을 사용하여 함수를 호출 할 수 있고
- "from module이름 import 함수명" 으로도 가능하다.

3. 모듈 사용하기

In [3]:
```
# 모듈의 함수 호출 1
import mod1
print(mod1.add(3,4))
```

```
7
```

In [4]:
```
# 모듈의 함수 호출 2
from mod1 import add
add(3, 4)
```

Out[4]:
```
7
```

In [5]:
```
# 콤마로 구분하여 필요한 함수 전부 불러오기
from mod1 import add, sub
```

In [6]:
```
sub(7,5)
```

Out[6]:
```
2
```

In [7]:
```
#모듈의 모든 함수 불러오기
from mod1 import *
sub(4,1)
```

Out[7]:
```
3
```

In [8]:
```
#모듈 내 변수 출력
import mod2
print(mod2.PI)
```

```
3.141592
```

In [9]:
```
#모듈 내 class 호출
a = mod2.Math()     #mod2.py 의 Math class 호출
print(a.solv(2))    #Math class 안의 solv 함수

#모듈 내 함수, 변수 호출
print(mod2.add(mod2.PI, 4.4)) #mod2.py의 add 함수 호출
```

```
12.566368
7.5415920000000005
```

Ch5-3. 내장 함수와 라이브러리

- 앞서 배운 함수와 클래스, 모듈을 사용하면 원하는 프로그램을 얼마든지 만들 수 있다.
- 물론 공부를 위해서라면 원하는 알고리즘을 실제로 구현해보는 연습을 하는 것이 의미 있지만 그런 목적이 아니라면 이미 만들어진 것을 다시 만드는 것은 불필요한 행동이다.우리
- 는 이렇게 충분한 검증을 거쳐 여러 사람들에게 사용되고 있는 몇 가지 프로그램 (내장 함수와 라이브러리)을 살펴 보고자 한다.

- 먼저, 내장 함수는 외부 모듈과 달리 import가 필요하지 않고 파이썬 배포본에 함께 들어있기 때문에 아무런 설정 없이 바로 사용할 수 있다.
- 파이썬 라이브러리는 전 세계의 파이썬 사용자들이 만든 유용한 프로그램을 모아 놓은 것으로 라이브러리를 설치한다음에 import 하여 사용하면 된다.

Table of Contents

1. 내장함수

- 우리는 이미 몇 가지 내장 함수를 배웠다. print, del, type 등이 바로 그것이다.
- 파이썬 내장 함수는 외부 모듈과 달리 import가 필요하지 않고 파이썬 배포본에 함께 들어있기 때문에 아무런 설정 없이 바로 사용할 수 있다.

이전에 배웠던 내장 함수와 함께 몇가지 추가로 자주 사용할만한 것들을 배워보자.

In [3]:
```
"""
abs(x) :절대값
"""
abs(-1.2)
```

Out[3]: 1.2

In [9]:
```
"""
all(x)
반복 가능한(iterable) 자료형 x를 인수로 받으며 이 x의 요소가 모두 참이면 True
거짓이 하나라도 있으면 False를 돌려준다.
"""
```

```
print( all([1,2,3]))      # 0이 아닌 숫자형 요소는 참이다.
print(all([1,2,0,3]))     #요소 0 은 거짓이다.
```

```
False
```

In [12]:

```
"""
any(x)
반복 가능한(iterable) 자료형 x를 입력 인수로 받으며 이 x의 요소 중 하나라도 참이 있으면 True를 돌려주고,
x가 모두 거짓일 때에만 False를 돌려준다.
"""
print(any([1,2,3,0]))   # 요소가 모두 거짓이 아니기 때문에 true를 돌려준다.
print(any([0, ""]))             # 요소 0과 ""은 모두 거짓이므로 false를 돌려 준다.
print(any([]))          # 입력 인수가 빈 값인 경우에는 False를 돌려준다.
```

```
True
False
False
```

In [18]:

```
"""
enumerate
함수는 순서가 있는 자료형(리스트, 튜플, 문자열)을 입력으로 받아
인덱스 값을 포함하는 enumerate 객체를 돌려준다.
※ 보통 enumerate 함수는 다음 예제처럼 for문과 함께 자주 사용한다.
"""
for i, name in enumerate(['body', 'foo', 'bar']):
    print(i, name)
```

```
0 body
1 foo
2 bar
```

In [24]:

```
"""
eval
실행 가능한 문자열(1+2, 'hi' + 'a' 같은 것)을 입력으로 받아 문자열을 실행한 결괏값을 돌려주는 함수이다.
보통 eval은 입력받은 문자열로 파이썬 함수나 클래스를 동적으로 실행하고 싶을 때 사용한다.
"""
print(eval('1+2'))
print(eval("'hi' + 'a'"))
print(eval('divmod(4, 3)'))
```

```
3
hia
(1, 1)
```

In [29]:

```
"""
id
id(object)는 객체를 입력받아 객체의 고유 주소 값(레퍼런스)을 돌려주는 함수
"""
a = 3
print(id(a))
b = a
print(id(b))
```

```
140732730941904
140732730941904
```

In [30]:
```
"""
input
input([prompt])은 사용자 입력을 받는 함수
"""
a = input()
```

```
hi
```

In [31]:
```
b = input("Enter: ")
```

```
Enter: python
```

In [32]:
```
b
```

Out[32]:
```
'python'
```

In [34]:
```
"""
int
int(x)는 문자열 형태의 숫자나 소수점이 있는 숫자 등을 정수 형태로 돌려주는 함수
x가 정수인 경우 그대로를 돌려준다.
"""
print(int('3'))
print(int(3.4))
```

```
3
3
```

In [35]:
```
"""
len
len(s)은 입력값 s의 길이(요소의 전체 개수)를 돌려주는 함수
"""
print(len("python"))
print(len([1,2,3]))
print(len((1,"a")))
```

```
6
3
2
```

In [43]:
```
"""
map
map(f, iterable)은 함수(f)와 반복 가능한(iterable) 자료형을 입력으로 받는다.
map은 입력받은 자료형의 각 요소를 함수 f가 수행한 결과를 묶어서 돌려주는 함수이다.
"""

# 만일 리스트 입 요소를 입력받아 각 요소에 2를 곱한 결과값을 돌려주고 싶은 경우라면,

def two_times(x):
    return x*2
```

```
list(map(two_times, [1, 2, 3, 4])) #map의 결과를 리스트로 보여 주기위해 list 함수를 사용하여 출력
```

Out[43]:
```
[2, 4, 6, 8]
```

In [44]:
```
#위의 예제를 lambda를 사용하면 아래와 같이 간략하게 표현 할 수 있다.
list(map(lambda a: a*2, [1, 2, 3, 4]))
```

Out[44]:
```
[2, 4, 6, 8]
```

In [47]:
```
"""
max
max(iterable)는 인수로 반복 가능한 자료형을 입력받아 그 최댓값을 돌려주는 함수
"""

"""
min
min(iterable)은 인수로 반복 가능한 자료형을 입력받아 그 최솟값을 돌려주는 함수
"""

print(max([1, 2, 3]))
print(max("python") )

print(min([1, 2, 3]))
print(min("python") )
```

```
3
y
1
h
```

In [66]:
```
"""
open
open(filename, [mode])은 "파일 이름"과 "읽기 방법"을 입력받아 파일 객체를 돌려주는 함수
"""

#w: 쓰기 모드로 파일 열기 (파일이 없으면 생성, 파일이 존재하면 파일 내용을 비움)
f=open("myfile.txt","w")
f.write('Open, read, write and close a file using Python')
f.close()

#r : 읽기 모드로 파일 열기
f = open("myfile.txt", "r")
f = open("myfile.txt") #mode는 r(읽기모드)이 default
f.read()
```

Out[66]:
```
'Open, read, write and close a file using Python'
```

In [67]:
```
f.close()
```

In [68]:
```
"""
pow
pow(x, y)는 x의 y 제곱한 결괏값을 돌려주는 함수
"""
```

```
pow(3, 3)
```

Out[68]: 27

In [70]:
```
"""
range
range([start,] stop [,step] )는 for문과 함께 자주 사용하는 함수
"""
print(list(range(5)))          # 0이상 5미만의 정수
print(list(range(5, 10)))      # 5이상 10미만의 정수
print(list(range(1, 10, 2))) #1부터 10미만까지 2 만큼의 간격의 숫자
```

```
[0, 1, 2, 3, 4]
[5, 6, 7, 8, 9]
[1, 3, 5, 7, 9]
```

In [72]:
```
"""
round
round(number[, ndigits]) 함수는 숫자를 입력받아 반올림해 주는 함수
round 함수의 두 번째 매개변수는 반올림하여 표시하고 싶은 소수점의 자릿수
"""
print(round(4.6))
print(round(5.678, 2)) # 소수점 2자리까지만 반올림하여 표시
```

```
5
5.68
```

In [74]:
```
"""
sorted
sorted(iterable) 함수는 입력값을 정렬한 후 그 결과를 리스트로 돌려주는 함수
"""
print(sorted(['a', 'c', 'b']))
print(sorted((3, 2, 1)))
print(sorted("zero"))
```

```
['a', 'b', 'c']
[1, 2, 3]
['e', 'o', 'r', 'z']
```

In [78]:
```
"""
str
str(object)은 문자열 형태로 객체를 변환하여 돌려주는 함수
"""
print(str(3))
print(str('hi'))
```

```
3
hi
```

In [82]:
```
"""
tuple
tuple(iterable)은 반복 가능한 자료형을 입력받아 튜플 형태로 바꾸어 돌려주는 함수
"""
print(tuple("abc"))
```

```
print(tuple([1, 2, 3]))
```

```
('a', 'b', 'c')
(1, 2, 3)
```

In [81]:
```
"""
list
list(s)는 반복 가능한 자료형 s를 입력받아 리스트로 만들어 돌려주는 함수
"""
print(list("python"))
print(list((1,2,3)))
```

```
['p', 'y', 't', 'h', 'o', 'n']
[1, 2, 3]
```

In [83]:
```
"""
type
type(object)은 입력값의 자료형이 무엇인지 알려 주는 함수
"""
print(type("abc"))
print(type([ ]))
```

```
<class 'str'>
<class 'list'>
```

In [84]:
```
"""
sum
sum(iterable) 은 입력받은 리스트나 튜플의 모든 요소의 합을 돌려주는 함수
"""
sum([1,2,3])
```

Out[84]: 6

In [85]:
```
"""
zip
zip(*iterable)은 동일한 개수로 이루어진 자료형을 묶어 주는 역할을 하는 함수
"""
list(zip([1, 2, 3], [4, 5, 6]))
```

Out[85]: [(1, 4), (2, 5), (3, 6)]

In [86]:
```
list(zip([1, 2, 3], [4, 5, 6], [7, 8, 9]))
```

Out[86]: [(1, 4, 7), (2, 5, 8), (3, 6, 9)]

In [87]:
```
list(zip("abc", "def"))
```

Out[87]: [('a', 'd'), ('b', 'e'), ('c', 'f')]

2. 라이브러리

- "라이브러리"는 "도서관"이라는 뜻 그대로 원하는 정보를 찾아보는 곳이다.
- 파이썬 라이브러리는 전 세계의 파이썬 사용자들이 만든 유용한 프로그램을 모아 놓은 것으로 패키지와 모듈의 집합이다.
- 파이썬 내장(표준) 라이브러리는 파이썬을 설치할 때 자동으로 컴퓨터에 설치되고, 외부라이브러리는 개발자가 필요에 의해 개발한 것 으로 설치하여 사용할 수 있다.
- (라이브러리를 설치한 다음에) import 하여 사용하면 된다.

이후로 우리는 Numpy, Pandas, Matplotlib 과 같은 외부 라이브러리를 설치하여 연산 및 분석에 활용하는 방법에 대해 학습해볼 예정이다.

Example

1. class 상속

Q1) 다음과 같은 Calculator 클래스가 있다. Calculator 클래스를 상속하여서, 객체변수 value가100 이상의 값은 가질 수 없도록 제한하는 MaxLimitCalculator 클래스를 만들어 보자.

In [5]:
```
# Calculator 클래스
class Calculator:
    def __init__(self):
        self.value = 0

    def add(self, val):
        self.value += val
```

다음은 MaxLimitCalculator의 실행 예시로써, 작성된 MaxLimitCalculator 은 다음과 같이 작동하게 된다.

In [8]:
```
cal = MaxLimitCalculator()
cal.add(50)  # +50
```

Out[8]:
```
50
```

In [9]:
```
print(cal.add(10))   # +10
```

```
60
```

In [10]:
```
print(cal.add(60))  # +60 (100 이상의 값은 100)
```

```
100
```

In [11]:
```
print(cal.add(20))   # +20 (100 이상의 값은 100)
```

```
100
```

2. Class 생성

Q1) 다음을 만족시키는 Date 클래스를 완성하여라.

is_date_valid는 입력된 문자열이 올바른 날짜인지 검사하는 메서드이다.
날짜에서 월은 12월까지 일은 31일까지 있어야 한다.

- 입력되는 날짜는 "년-월-일" 형식으로 입력됨.
- is_date_valid 매서드는 입력된 날짜 "년-월-일" 중에서 "월"이 1부터 12의 값을 갖고, "일"은 1부터 31의 값을 갖으면 "올바른 날짜 형식입니다"를 출력하고, 그렇지 않은 경우 "잘못된 날짜 형식입니다"를 출력한다.

In [ ]:
```
class Date:
    def __init__ ~~
    def is_date_valid ~~
```

3. 내장함수와 모듈

Q1) 다음 결과를 예측해 보자.

In [ ]:
```
all([1, 2, abs(-3)-3, abs(-3)-4])
```

In [ ]:
```
any([1, 2, abs(-3)-3, abs(-3)-4])
```

Q2) map과 lambda를 사용하여 [1, 2, 3, 4] 리스트의 각 요솟값에 4가 곱해진 리스트 [4, 8, 12, 16]를 만들어 보자.

Q3) [-8, 2, 7, 5, -3, 5, 0, 1] 리스트의 리스트의 최댓값과 최솟값의 합을 구하여라.

In [92]:
```
a=[-8, 2, 7, 5, -3, 5, 0, 1]
```

Q4) map과 lambda를 사용하여 리스트 a에서 3의 배수를 문자열로 변하여 [1, 2, '3', 4, 5, '6', 7, 8, '9', 10]를 만들어 보자.

In [ ]:
```
a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

Q5) time모듈은 현재의 날짜와 시간을 출력해주는 strftime 매서드가 있다. time 모듈의 strftime 매서드를 활용해서 현재의 날짜와 시간을 다음과 같은 형식으로 출력해보자.

- strftime 매서드는 날짜와 시간 출력의 형식을 지정하는 argument가 존재하며, 아래와 같은 형식을 위해서는 argument 를 "%Y/%m/%d %H:%M:%S"로 하여서 만들어 낼 수 있다.

2022/08/30 11:18:34