

# 12. 구조체와 파이썬클래스

- 파이썬 built-in 자료형
- 구조체 개념
- 클래스 기본
- 속성과 메소드
- 생성자와 소멸자
- 연산자 중복
- built-in 자료형과 객체

# 파이썬 built\_in 자료형 (Data Types)

- Numeric 자료형 : int, float, complex
- Sequence 자료형 : list, tuple, str
- Set 자료형 : set
- Mapping 자료형 : dict

```
>>> a = 10
>>> type(a)
<class 'int'>
>>> b = 3.5
>>> type(b)
<class 'float'>
>>> c = 3 + 4j
>>> type(c)
<class 'complex'>
```

```
>>> x = [1,2,3,4,5]
>>> type(x)
<class 'list'>
>>> y = (1,3,5,7,9)
>>> type(y)
<class 'tuple'>
>>> t = 'python programming'
>>> type(t)
<class 'str'>
```

```
>>> s = {1,2,3,4,5}
>>> type(s)
<class 'set'>
>>> d = {'1':'one', '2':'two', '3':'three'}
>>> type(d)
<class 'dict'>
```

# 구조체 개념

## ◆ 구조체

- 구조체는 데이터 여러 개를 묶어서 하나의 데이터를 표현할 수 있는 방법을 제공한다.
- 프로그래머가 원하는 대로 자료형을 만들 수 있도록 하는 기능이다.
- 구조체의 예

학생을 표현하는  
구조체

| name | ban | kor | math | eng |
|------|-----|-----|------|-----|
| 홍길동  | 1   | 90  | 85   | 93  |

좌표를 표현하는  
구조체

| x | y |
|---|---|
| 2 | 5 |

# 구조체 개념

## ◆ 구조체 정의

- 구조체는 구조체를 의미하는 **STRUCTURE** 키워드를 이용하여 정의한다.  
(IT개론 수업에서는 **struct** 라고 사용한다.)
- **struct** 키워드 다음에 구조체 이름이 오며 중괄호 { } 사이에 구조체에서 사용되는 변수 이름을 콤마로 구분하면서 나열한다.
- 구조체 안에 포함된 변수들을 '구조체의 멤버 변수'라고 한다.

순서도에서는 STRUCTURE 키워드 이용

|     |      |     |     |      |     |
|-----|------|-----|-----|------|-----|
| 구조체 | name | ban | kor | math | eng |
|     | 홍길동  | 1   | 90  | 85   | 93  |

**struct** **score** { name; ban; kor; math; eng }

구조체이름

구조체 멤버 변수 이름

# 구조체 개념

## ◆ 구조체 정의와 구조체 변수 선언

```
struct score { name; ban; kor; math; eng }
```

구조체 score 변수명

```
struct score {  
    name;  
    ban;  
    kor;  
    math;  
    eng;  
}
```

```
struct score scr ;  
scr.name = "홍길동";  
scr.ban = 1;  
scr.kor = 90;  
scr.math = 85;  
scr.eng = 93;
```

scr →

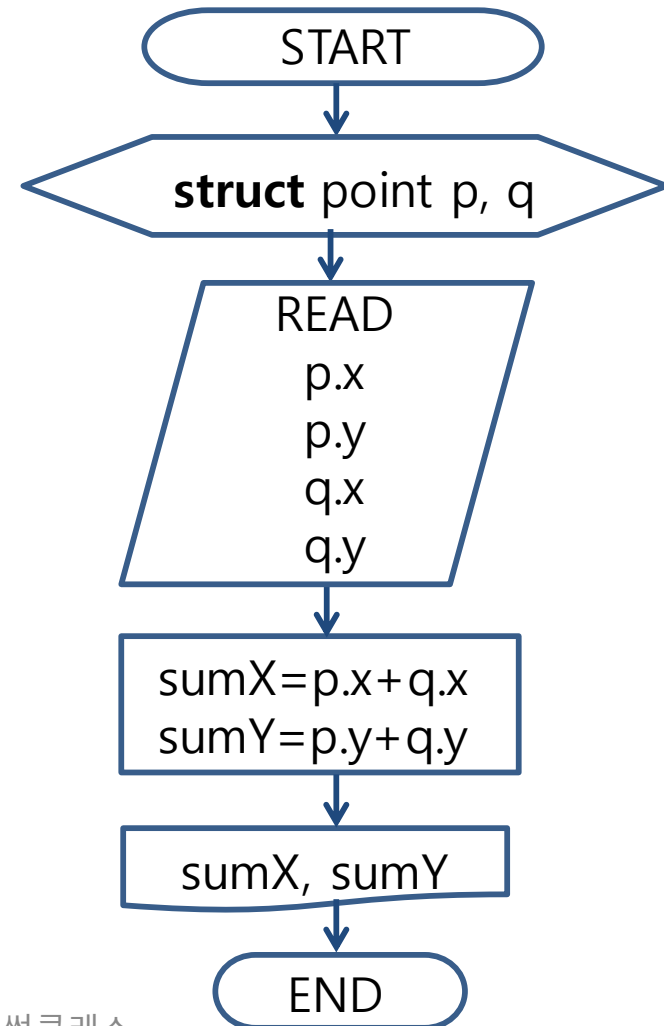
| name | ban | kor | math | eng |
|------|-----|-----|------|-----|
| 홍길동  | 1   | 90  | 85   | 93  |

**instance(인스턴스)** 또는 **object(객체)**라고 부른다.

# 구조체 예제

- 좌표평면상의 점을 나타내는 구조체를 정의하고 두 점 p와 q의 좌표를 읽어들이어 두 좌표의 합을 출력하는 순서도

```
struct point {  
    x;           ← x좌표  
    y;           ← y좌표  
}
```



# 클래스 기본

---

- 클래스를 정의하는 것은 구조체와 같이 새로운 자료형을 만드는 것이고, 인스턴스는 이 자료형의 객체를 생성하는 것이다.
- 클래스 구성 요소
  - 멤버데이터 (member data) : 속성(attribute)이라고도 함.
  - 메소드 (method)
  - 생성자 (constructor), 소멸자 (destructor)
  - 연산자 중복 (operator overloading)
- 클래스 정의와 선언
  - 클래스는 선언과 동시에 클래스 객체가 생성됨.
  - 클래스 선언을 통해 새로운 이름 공간이 생성됨.
  - 파이썬에서는 멤버 변수, 멤버 메소드의 접근 권한은 **public**임.
  - self : C++, 자바의 this 키워드에 해당함.

# 클래스 기본

클래스명

**class** simple: # 헤더 (header)

# 몸체 (body)

← 멤버데이터, 메소드,  
생성자, 소멸자,  
연산자 중복

- 메소드, 생성자, 소멸자, 연산자 중복은 함수와 유사한 형태이다.
- 멤버데이터(속성)만으로 구성될 수도 있다.
- 메소드만으로 구성될 수도 있다.

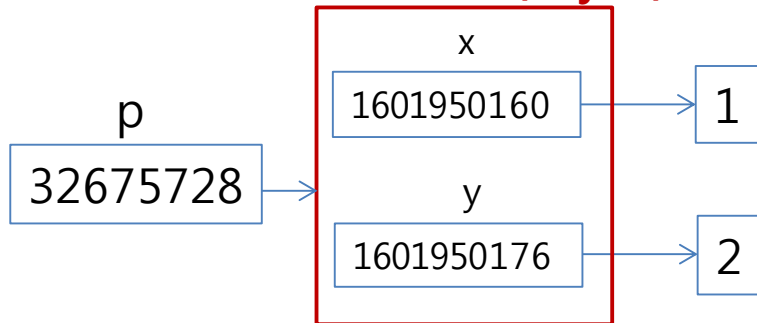


# 클래스 기본

## ■ 속성만을 갖는 클래스 예

```
class Point:  
    x = 1  
    y = 2
```

**instance (object)**



```
>>> dir(Point)  
['_class_', '__delattr__', '__dict__', '__dir__',  
 '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattr__', '__gt__', '__hash__',  
 '__init__', '__le__', '__lt__', '__module__',  
 '__ne__', '__new__', '__reduce__',  
 '__reduce_ex__', '__repr__', '__setattr__',  
 '__sizeof__', '__str__', '__subclasshook__',  
 '__weakref__', 'x', 'y']
```

```
>>> p = Point() # 인스턴스 생성하기  
>>> p.x  
1  
>>> p.y  
2  
>>> type(Point)  
<class 'type'>  
>>> type(p)  
<class '__main__.Point'>  
>>> Point  
<class '__main__.Point'>  
>>> p  
<__main__.Point object at 0x01F29790>  
>>> id(p)  
32675728  
>>> id(Point) # 클래스 자체를 객체로 취급  
33302768  
>>> id(p.x)  
1601950160  
>>> id(p.y)  
1601950176
```

# 클래스 기본

- 파이썬에서는 이미 정의된 클래스에 속성을 추가할 수 있다.

```
class Data:  
    a = 100
```

```
>>> d = Data()  
>>> d.a  
100  
>>> dir(d)  
['_class_', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',  
 '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__',  
 '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'a']  
>>> d.b = 200 # 속성 'b'를 추가  
>>> d.b  
200  
>>> dir(d)  
['_class_', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__',  
 '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__', '__le__', '__lt__',  
 '__module__', '__ne__', '__new__', '__reduce__', '__reduce_ex__', '__repr__',  
 '__setattr__', '__sizeof__', '__str__', '__subclasshook__', '__weakref__', 'a', 'b']  
>>>
```

# 클래스 기본

- **멤버데이터(속성)**과 **메소드**를 갖는 클래스

```
class Point:
```

```
    x = 0    # 멤버 데이터
```

```
    y = 0    # 멤버 데이터
```

```
    def inc(self, a, b):    # 메소드의 첫 번째 인자는 반드시 self여야 함.  
        self.x += a        # self를 반드시 붙여야 함.
```

```
        self.y += b
```

```
    def dec(self, a, b):    # 메소드
```

```
        self.x -= a
```

```
        self.y -= b
```

```
    def printPoint(self):    # 메소드
```

```
        print('x:', self.x, ', y:', self.y)
```

```
>>> p = Point()
>>> p.printPoint()
x : 0 , y : 0
>>> p.inc(5,7)
>>> p.printPoint()
x : 5 , y : 7
>>> p.dec(2,9)
>>> p.printPoint()
x : 3 , y : -2
```

```
class Point:
    x = 0
    y = 0
    def inc(self, a, b):
        self.x += a
        self.y += b
    def dec(self, a, b):
        self.x -= a
        self.y -= b
    def printPoint(self):
        print('x:', self.x, ', y:', self.y)
```

하나의 파일에 class만 저장하고 IDLE에서 다음과 같이 수행한다.

```
>>> p = Point()
>>> p.printPoint()
x : 0 , y : 0
>>> p.inc(10,20)
>>> p.printPoint()
x : 10 , y : 20
>>> p2 = Point()
>>> p2.printPoint()
x : 0 , y : 0
>>> p2.inc(20,30)
>>> p2.printPoint()
x : 20 , y : 30
>>> p2.dec(5,7)
>>> p2.printPoint()
x : 15 , y : 23
>>> p.x
10
>>> p2.y
23
```

```
class Point:
    x = 0
    y = 0
    def inc(self, a, b):
        self.x += a
        self.y += b
    def dec(self, a, b):
        self.x -= a
        self.y -= b
    def printPoint(self):
        print('x :', self.x, ', y :', self.y)
```

```
if __name__ == '__main__':
    p = Point()
    p.printPoint()
    p.inc(10,20)
    p.printPoint()

    p2 = Point()
    p2.printPoint()
    p2.inc(20,30)
    p2.dec(5,7)
    p2.printPoint()
```

하나의 파일에 class와 main 부분을 모두 입력하고 수행한다.


```
x : 0 , y : 0
x : 10 , y : 20
x : 0 , y : 0
x : 15 , y : 23
```

# 클래스 기본


- 메소드 인수 부분에 self를 넣지 않는다면...

```
class Point:
    x = 0
    y = 0
    def inc(a, b): # 첫 인자 self 삭제
        self.x += a
        self.y += b
    def printPoint(self):
        print('x :', self.x, ', y :', self.y)
```

```
class Point:
    x = 0
    y = 0
    def inc(self, a, b):
        x += a # self를 제거함
        self.y += b
    def printPoint(self):
        print('x :', self.x, ', y :', self.y)
```



```
>>> p = Point()
>>> p.printPoint()
x : 0 , y : 0
>>> p.inc(1,2)
Traceback (most recent call last):
  File "<pyshell#88>", line 1, in <module>
    p.inc(1,2)
TypeError: inc() takes 2 positional arguments but 3 were given
```



```
>>> p = Point()
>>> p.printPoint()
x : 0 , y : 0
>>> p.inc(1,2)
Traceback (most recent call last):
  File "<pyshell#91>", line 1, in <module>
    p.inc(1,2)
  File "C:/Python34/funtest.py", line 5, in inc
    x += a
UnboundLocalError: local variable 'x' referenced before assignment
```

# 클래스 기본

- Person 클래스 예 (이름과 나이로 Person을 표현하는 클래스)

```
class Person:
    name = "default"      # 멤버 데이터
    age = 10              # 멤버 데이터
    def incAge(self, inc): # 메소드
        self.age += inc
    def decAge(self, dec): # 메소드
        self.age -= dec
    def printData(self):   # 메소드
        print("name :", self.name, ", age :", self.age)
```

```
>>> p1 = Person()
>>> p1.name = "Alice"
>>> p1.age = 15
>>> p1.printData()
name : Alice , age : 15
```

```
>>> p1 = Person()
>>> p1.printData()
name : default , age : 10
```

```
>>> p2 = Person()
>>> p2.name = "Bob"
>>> p2.age = 12
>>> p2.printData()
name : Bob , age : 12
>>> p2.incAge(2)
>>> p2.printData()
name : Bob , age : 14
```

# 클래스 기본

```
class data:
    a = 10
    def printA(self):
        print("a :", self.a)
    def printB(self):
        print("b :", self.b)
```

```
>>> d1 = data()
```

```
>>> d1.printA()
```

```
a : 10
```

```
>>> d1.printB()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#84>", line 1, in <module>
```

```
d1.printB()
```

```
File "<pyshell#81>", line 6, in printB
```

```
print("b :", self.b)
```

**AttributeError:** 'data' object has no attribute 'b' (현재 멤버데이터에 b는 없다)

```
>>> d1.b = 100 # 멤버데이터 추가 후에는 printB 호출 가능하다
```

```
>>> d1.printB()
```

```
b : 100
```

없는 메소드를 호출해도 AttributeError 발생

```
>>> d = data()
```

```
>>> d.printC()
```

```
Traceback (most recent call last):
```

```
File "<pyshell#14>", line 1, in <module>
```

```
d.printC()
```

**AttributeError:** 'data' object has no attribute 'printC'



# 메소드 정의와 호출

---

```
>>> class MyClass:
    def set(self, v):
        self.value = v
    def get(self):
        return self.value
```

```
>>> m = MyClass()
>>> n = MyClass()
>>> m.set(10)
>>> MyClass.set(n,20)
>>> MyClass.get(m)
10
>>> n.get()
20
```

'인스턴스.메소드' 호출 시에는  
self 인자는 쓰지 않음.

'클래스.메소드' 호출 시에는 **self** 위치에  
인스턴스를 명시해야 함.

# 메소드 정의와 호출

## ■ 생성자와 소멸자

- 생성자(constructor) : 인스턴스 객체가 생성될 때 초기화를 위해서 자동으로 호출되는 초기화 메소드.
- 소멸자(destructor) : 인스턴스 객체를 사용하고서 메모리에서 제거할 때 자동으로 호출되는 메소드.

생성자 함수의 이름 **`__init__`**  
소멸자 함수의 이름 **`__del__`**

```
class MyClass2:
    def __init__(self):    # 인스턴스 생성시에 자동으로 호출
        self.value = 0
    def set(self, v):
        self.value = v
    def get(self):
        return self.value
```

```
>>> c = MyClass2()
>>> c.get()
0
```

# 메소드 정의와 호출

---

```
class MyClass2:
    def __init__(self):
        self.value = 100
        print("created with value :", self.value)
    def get(self):
        return self.value
    def __del__(self):    # 객체 소멸시에 자동으로 호출
        print("deleted with value :", self.value)
    def set(self, v):
        self.value = v
```

```
>>> x = MyClass2()
created with value: 100
>>> x.set(200)
>>> x.get()
200
>>> del x
deleted with value: 200
```

# 메소드 정의와 호출

```
class Person:
    def __init__(self, name, age):    # 생성자
        self.name = name
        self.age = age
    def incAge(self, inc):
        self.age += inc
    def decAge(self, dec):
        self.age -= dec
    def printData(self):
        print("name :", self.name, ", age :", self.age)

if __name__ == '__main__':
    p1 = Person('Alice', 10)    # 생성자 자동 호출
    p2 = Person('Bob', 8)      # 생성자 자동 호출
    p1.printData()
    p2.printData()
```

```
>>> type(Person)
<class 'type'>
>>> type(p1)
<class '__main__.Person'>
>>> type(p2)
<class '__main__.Person'>
```

int가 클래스임을 알 수 있다.

```
>>> x = 10
>>> type(x)
<class 'int'>
>>> y = int(20)
>>> type(y)
<class 'int'>
>>> type(int)
<class 'type'>
>>>
```

# 클래스 기본 구성

---

- 클래스를 만들어서 클래스의 구성 내용을 살펴 본다.

```
class Basic:  
    x = 10  
    def __printBasic__(self):  
        print('x :', x)
```

```
>>> dir(Basic)  
['_class_', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__', '__format__', '__ge__',  
 '__getattr__', '__gt__', '__hash__', '__init__', '__le__', '__lt__', '__module__', '__ne__',  
 '__new__', __printBasic__, '__reduce__', '__reduce_ex__', '__repr__', '__setattr__',  
 '__sizeof__', '__str__', '__subclasshook__', '__weakref__', x]
```

# 클래스 기본 구성

```
class NewType:
    def __init__(self, data):
        self.data = data
    def __add__(self, x):
        return self.data + x
    def __sub__(self, x):
        return self.data - x
    def __mul__(self, x):
        return self.data * x
    def __truediv__(self, x):
        return self.data / x
    def __floordiv__(self, x):
        return self.data // x
    def __mod__(self, x):
        return self.data % x
    def __divmod__(self, x):
        return (self.data//x, self.data%x)
    def __pow__(self, x):
        return self.data **x
```

```
>>> n = NewType(20)
>>> n.__add__(5)
25
>>> n.__sub__(7)
13
>>> n.__mul__(3)
60
>>> n.__truediv__(6)
3.3333333333333335
>>> n.__floordiv__(6)
3
>>> n.__mod__(6)
2
>>> n.__divmod__(6)
(3, 2)
>>> n.__pow__(3)
8000
```

```
>>> m = NewType(20)
>>> m + 5
25
>>> m - 7
13
>>> m * 3
60
>>> m / 6
3.3333333333333335
>>> m // 6
3
>>> m % 6
2
>>> divmod(m,6)
(3, 2)
>>> m ** 3
8000
```

# 연산자 중복 (operator overloading)

- 클래스는 연산자 중복을 지원한다.
- 파이썬에서 사용하는 모든 연산자(각종 산술, 논리 연산자, 슬라이싱, 인덱싱 등)의 동작을 직접 정의할 수 있다.
- 연산자를 중복하면 내장 자료형과 비슷한 방식으로 동작하는 클래스를 설계할 수 있다.

```
class MyClass:  
    def __add__(self, x): # 이름 __add__는 + 연산자를 중복한다.  
        print('add {} called'.format(x))  
        return x
```

```
>>> a = MyClass()  
>>> a + 3          # 더하기 연산자와 중복  
add 3 called  
3
```

# 연산자 중복 (operator overloading)

- 수치 연산자 메소드

| 메소드   | 연산자                                |
|---|------------------------------------|
| <code>__add__(self, other)</code>           | <code>+</code>                     |
| <code>__sub__(self, other)</code>           | <code>-</code>                     |
| <code>__mul__(self, other)</code>           | <code>*</code>                     |
| <code>__truediv__(self, other)</code>       | <code>/</code>                     |
| <code>__floordiv__(self, other)</code>      | <code>//</code>                    |
| <code>__mod__(self, other)</code>           | <code>%</code>                     |
| <code>__divmod__(self, other)</code>        | <code>divmod()</code>              |
| <code>__pow__(self, other[, modulo])</code> | <code>pow()</code> <code>**</code> |
| <code>__lshift__(self, other)</code>        | <code>&lt; &lt;</code>             |
| <code>__rshift__(self, other)</code>        | <code>&gt; &gt;</code>             |
| <code>__and__(self, other)</code>           | <code>&amp;</code>                 |
| <code>__xor__(self, other)</code>           | <code>^</code>                     |
| <code>__or__(self, other)</code>            | <code> </code>                     |



# 연산자 중복 (operator overloading)

class Point:

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

```
    self.x = x
```

```
    self.y = y
```

```
def set(self, x, y):
```

```
    self.x = x
```

```
    self.y = y
```

```
def get(self):
```

```
    return "(" + str(self.x) + "," + str(self.y) + ")"
```

```
def __add__(self, other):
```

```
    newX = self.x + other.x
```

```
    newY = self.y + other.y
```

```
    return Point(newX, newY)
```

두 좌표의 합 구하기

```
>>> p1 = Point(2,3)
>>> p2 = Point(4,7)
>>> p3 = p1 + p2
>>> p1.get()
'(2,3)'
>>> p2.get()
'(4,7)'
>>> p3.get()
'(6,10)'
```

# 연산자 중복 (operator overloading)

- 피연산자가 바뀐 경우의 수치 연산자 메소드

| 메소드  | 연산자                                |
|--|------------------------------------|
| <code>__radd__(self, other)</code>           | <code>+</code>                     |
| <code>__rsub__(self, other)</code>           | <code>-</code>                     |
| <code>__rmul__(self, other)</code>           | <code>*</code>                     |
| <code>__rtruediv__(self, other)</code>       | <code>/</code>                     |
| <code>__rfloordiv__(self, other)</code>      | <code>//</code>                    |
| <code>__rmod__(self, other)</code>           | <code>%</code>                     |
| <code>__rdivmod__(self, other)</code>        | <code>divmod()</code>              |
| <code>__rpow__(self, other[, modulo])</code> | <code>pow()</code> <code>**</code> |
| <code>__rlshift__(self, other)</code>        | <code>&lt; &lt;</code>             |
| <code>__rrshift__(self, other)</code>        | <code>&gt; &gt;</code>             |
| <code>__rand__(self, other)</code>           | <code>&amp;</code>                 |
| <code>__rxor__(self, other)</code>           | <code>^</code>                     |
| <code>__ror__(self, other)</code>            | <code> </code>                     |

# 연산자 중복 (operator overloading)

```
class NewType2:
    def __init__(self, data):
        self.data = data
    def __radd__(self, x):
        return x + self.data
    def __rsub__(self, x):
        return x - self.data
    def __rmul__(self, x):
        return x * self.data
    def __rtruediv__(self, x):
        return x / self.data
    def __rfloordiv__(self, x):
        return x // self.data
    def __rmod__(self, x):
        return x % self.data
    def __rdivmod__(self, x):
        return (x // self.data, x % self.data)
    def __rpow__(self, x):
        return x ** self.data
```

```
>>> n = NewType2(20)
>>> 10 + n
30
>>> 33 - n
13
>>> 2 * n
40
>>> 37 / n
1.85
>>> 37 // n
1
>>> 37 % n
17
>>> divmod(37, n)
(1, 17)
>>> 2 ** n
1048576
```

# 연산자 중복 (operator overloading)

- 수치 연산자 : 확장 산술 연산자 `s += b → s.__iadd__(b)`

| 메소드  | 연산자                    |
|--|------------------------|
| <code>__iadd__(self, other)</code>           | <code>+=</code>        |
| <code>__isub__(self, other)</code>           | <code>-=</code>        |
| <code>__imul__(self, other)</code>           | <code>*=</code>        |
| <code>__itruediv__(self, other)</code>       | <code>/=</code>        |
| <code>__ifloordiv__(self, other)</code>      | <code>//=</code>       |
| <code>__imod__(self, other)</code>           | <code>%=</code>        |
| <code>__ipow__(self, other[, modulo])</code> | <code>**=</code>       |
| <code>__ilshift__(self, other)</code>        | <code>&lt;&lt;=</code> |
| <code>__irshift__(self, other)</code>        | <code>&gt;&gt;=</code> |
| <code>__iand__(self, other)</code>           | <code>&amp;=</code>    |
| <code>__ixor__(self, other)</code>           | <code>^=</code>        |
| <code>__ior__(self, other)</code>            | <code> =</code>        |

# 연산자 중복 (operator overloading)

```
class NewType3:
    def __init__(self, data):
        self.data = data
    def __iadd__(self, x):
        self.data += x
        return self.data
    def __isub__(self, x):
        self.data -= x
        return self.data
    def __imul__(self, x):
        self.data *= x
        return self.data
    def __itruediv__(self, x):
        self.data /= x
        return self.data
    def __ifloordiv__(self, x):
        self.data //= x
        return self.data
    def __imod__(self, x):
        self.data %= x
        return self.data
    def __ipow__(self, x):
        self.data **= x
        return self.data
```

```
>>> n = NewType3(30)
>>> n += 10
>>> print(n)
40
>>> n -= 10
>>> print(n)
30
>>> n *= 2
>>> print(n)
60
>>> n /= 7
>>> print(n)
8.571428571428571
>>> n = NewType3(50)
>>> n //= 9
>>> print(n)
5
>>> n %= 3
>>> print(n)
2
>>> n **= 10
>>> print(n)
1024
```

# 연산자 중복 (operator overloading)

---

- 비교 연산을 위한 메소드

| 연산자 | 메소드                        |
|-----|----------------------------|
| <   | object.__lt__(self, other) |
| <=  | object.__le__(self, other) |
| >   | object.__gt__(self, other) |
| >=  | object.__ge__(self, other) |
| ==  | object.__eq__(self, other) |
| !=  | object.__ne__(self, other) |

# 연산자 중복 (operator overloading)

## ■ 비교 연산을 위한 메소드

```
class Compare:
    def __init__(self, n):
        self.n = n
    def __eq__(self, o):
        print('__eq__ called')
        return self.n == o
    def __lt__(self, o):
        print('__lt__ called')
        return self.n < o
    def __le__(self, o):
        print('__le__ called')
        return self.n <= o
```

```
>>> c = Compare(10)
```

```
>>> c.__lt__(5)
```

```
__lt__ called
```

```
False
```

```
>>> c < 5
```

```
__lt__ called
```

```
False
```

```
>>> c.__lt__(20)
```

```
__lt__ called
```

```
True
```

```
>>> c < 20
```

```
__lt__ called
```

```
True
```

```
>>> c.__gt__(2)
```

```
NotImplemented
```

```
>>> c > 2
```

```
Traceback (most recent call last):
```

```
File "<pyshell#6>", line 1, in <module>
```

```
c > 2
```

```
TypeError: unorderable types: Compare() > int()
```

```
>>> c == 10
```

```
__eq__ called
```

```
True
```

```
>>> c != 10
```

```
__eq__ called
```

```
False
```

# 연산자 중복 (operator overloading)

- 문자열 변환 연산 : `__str__()` , `__repr__()` 메소드

| 문자열에서 str, repr 사용  | 다른 자료형에서 str, repr 사용   |
|---|---|
| <pre>&gt;&gt;&gt; s = 'hello' &gt;&gt;&gt; str(s) 'hello' &gt;&gt;&gt; repr(s) "'hello'" &gt;&gt;&gt; print(s) hello &gt;&gt;&gt; str(s) == repr(s) False</pre> | <pre>&gt;&gt;&gt; a = 100 # 기본 자료형 &gt;&gt;&gt; print(a) 100 &gt;&gt;&gt; str(a) '100' &gt;&gt;&gt; repr(a) '100' &gt;&gt;&gt; str(a) == repr(a) True &gt;&gt;&gt; list = [1,2,3,4,5] # 리스트 &gt;&gt;&gt; print(list) [1, 2, 3, 4, 5] &gt;&gt;&gt; str(list) '[1, 2, 3, 4, 5]' &gt;&gt;&gt; repr(list) '[1, 2, 3, 4, 5]' &gt;&gt;&gt; str(list) == repr(list) True</pre> |



# 연산자 중복 (operator overloading)

- 문자열 변환 연산 : `__str__()` , `__repr__()` 메소드

```
class StrRepr:
    def test(self):
        print('test')
```

```
>>> s = StrRepr()
>>> s.test()
test
>>> print(s)
<__main__.StrRepr object at 0x01F29790>
>>> str(s)
'<__main__.StrRepr object at 0x01F29790>'
>>> repr(s)
'<__main__.StrRepr object at 0x01F29790>'
```

```
class StrRepr:
    def test(self):
        print('test')
    def __str__(self):
        return 'str called'
```

```
>>> t = StrRepr()
>>> print(t)
str called
>>> str(t)
'str called'
```

print 함수는 `__str__` 메소드를 호출한다.

# 연산자 중복 (operator overloading)

- 문자열 변환 연산 : `__str__()` , `__repr__()` 메소드

```
class StrRepr:
    def test(self):
        print('test')
    def __str__(self):
        return 'str called'
    def __repr__(self):
        return 'repr called'
```

```
>>> t = StrRepr()
>>> print(t)
str called
>>> str(t)
'str called'
>>> repr(t)
'repr called'
```

```
class StrRepr:
    def test(self):
        print('test')
    def __repr__(self):
        return 'repr called'
```

```
>>> t = StrRepr()
>>> print(t)
repr called
>>> str(t)
'repr called'
>>> repr(t)
'repr called'
```

print 함수는 `__str__` 메소드가 없으면 `__repr__` 메소드를 호출한다.

# Built\_in 자료형과 객체

```
>>> a = 100
```

```
>>> type(a)
```

```
<class 'int'>
```

```
>>> b = int(50)
```

```
>>> type(b)
```

```
<class 'int'>
```

```
>>> dir(a)
```

```
['_abs_', '__add__', '__and__', '__bool__', '__ceil__', '__class__', '__delattr__', '__dir__',  
'_divmod_', '__doc__', '__eq__', '__float__', '__floor__', '__floordiv__', '__format__', '__ge__',  
'_getattribute__', '__getnewargs__', '__gt__', '__hash__', '__index__', '__init__', '__int__',  
'_invert_', '__le__', '__lshift__', '__lt__', '__mod__', '__mul__', '__ne__', '__neg__', '__new__',  
'_or_', '__pos__', '__pow__', '__radd__', '__rand__', '__rdivmod__', '__reduce__', '__reduce_ex__',  
'_repr_', '__rfloordiv__', '__rlshift__', '__rmod__', '__rmul__', '__ror__', '__round__', '__rpow__',  
'_rrshift_', '__rshift_', '__rsub_', '__rtruediv__', '__rxor__', '__setattr__', '__sizeof__', '__str__',  
'_sub_', '__subclasshook__', '__truediv__', '__trunc__', '__xor__', 'bit_length', 'conjugate',  
'denominator', 'from_bytes', 'imag', 'numerator', 'real', 'to_bytes']
```

```
>>> a.__add__(b)
```

```
150
```

```
>>> a.__ge__(b)
```

```
True
```

```
>>> a.__lt__(b)
```

```
False
```

```
>>> a+b
```

```
150
```

```
>>> a >= b
```

```
True
```

```
>>> a < b
```

```
False
```

## 클래스 int의 메소드