파이썬 기초
프로그래밍

함수

- **1**부터 실 인수 까지 중 짝수 만 출력

```
def evenprint(n, step = 1):
        x = 1
        while x <= n:
                if x%2 == 0:
                        print(x)
                x += step

evenprint(10)
```

- 가변 인수 활용 **(** 튜플**,** 사전 **)**

```
def argsfunct(*args):
        i = 0
        for x in args:
                i += 1
        print("args 인수의 개수 : %d" %i)
        print(args)
        print(args[0])
        print(args[1])
        print(args[2])

def dictsfunct(**dicts):
        i = 0
        for x in dicts.keys():
                i += 1
        print("dicts 인수의 개수 : %d" %i)
        print(dicts)

argsfunct(1,[2,4],{'a':1,'b':4,'c':5})
dictsfunct(a=1,b=2,c=3)
```

함수

- 튜플, 사전, 튜플 리스트 처리 함수

```
def func3(myarg):
        for num , value in myarg:
                print(num, value)


def func2(myarg):
        for num  in myarg:
                print(num, myarg[num])

def func1(myarg):
        for num  in myarg:
                print(num)


mydata = (1,2,3,4,5)
mydata2 = { 'A':1, 'B':2, 'C': 3}
mydata3 = [ (1,2), (6,8), (9, 3)]

func1(mydata)
func2(mydata2)
func3(mydata3)
```

함수

- 가변 인수로 받은 데이터에 실 인수 값의 곱한 내용을 출력

```
def mysum_func(mul, *args):
        cnt = 0
        tuple_len = len(args)
        while cnt < tuple_len:
                print("{}".format(mul * args[cnt]))
                cnt += 1

res = mysum_func(5, 1,2,3,4,5)
```

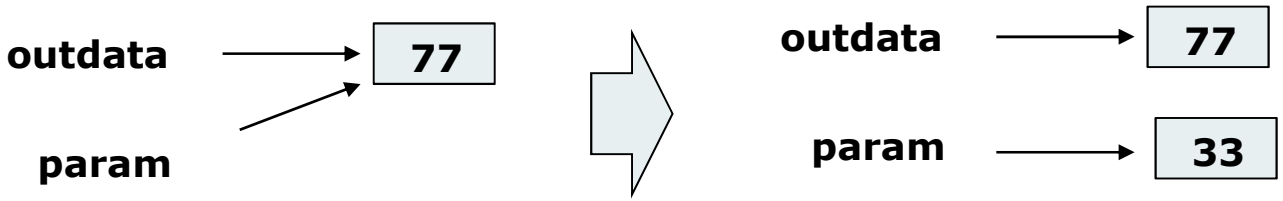
- 알파벳 문자열을 실 인수로 전달하여 문자열을 모두 대문자로 변환하는 함수

```
def conv_string_upper(mystr):
        convstr = mystr.upper()     // 대문자 변환 메소드
        print(convstr)

conv_string_upper("python")
```

함수

- 인수 전달**(id복사) //** 전역변수 값 수정 불가

```
outdata = 77
def func(param):
        param = 33
        print(param)
        print(locals())

func(outdata)
print(outdata)
print(locals())
```



- 전역 변수 값 수정하기 위한 리턴 값 활용

```
outdata = 77
def func(param):
        param = 33
        print("local var : ", param)
        return param

print("global var : ", outdata)
outdata = func(outdata)
print("return var : ", outdata)
```

- 전역 변수 값 수정하기 위한 **global** 키워드 활용

```
outdata = 77
def func():
        global outdata
        outdata = 50

print("global var : ", outdata)
func()
print("modify global var : ", outdata)
```

함수

- 전역 변수 값 수정하기 위한 **list** 활용

```
outdatalist = [30]
def func(param):
        param[0] = param[0] + 50

print(outdatalist[0])
func(outdatalist)
print(outdatalist[0])
```

- 람다 표현식

이름 없는 함수 표현식

lambda 인수 **:** 표현식

표현식 **=> x = 2** 와 같은 구문이 올 수 없고 리턴값을 취하는 표현식이 와야 한다**.**

```
my_list = [lambda x : x**2, lambda x : x+5 ]
print(my_list)
res = my_list[1](5)
print(res)
```

- 클로저 활용해 함수간 공유되는 전역변수 문제 해결

// 함수에서 사용되는 변수와 함수 정의를 내포하는 함수를 구현하여 이름 공간을 구별해 사용

```
val = 0
def outlinefunc():
        val = 77
        def innerlinefunc():
                nonlocal val
                val += 1
                print(val)
        return innerlinefunc
```

```
myclosure1 = outlinefunc()
myclosure2 = outlinefunc()

myclosure1()
myclosure1()

myclosure2()
myclosure2()
```

각각의 클로저

클로저로 묶여서 반환된 지역변수 **val**은 함수 종료후에도 사라지지 않음

함수

- 전달인자에 따른 여러 형태의 함수 함수

```
def tuple_args_func(num, *args):
        total = 0
        for i in args:
                print("i : %d" %i)
                total += i*num
        print(total)

def convstring(string):
        up_str = string.upper()
        print(up_str)

def default_arg_func(num1, num2 = 10):
        res = num1 * num2
        return res

tuple_args_func(3, 1,2,3,4,5)

convstring("mdsacademy")

value = default_arg_func(3)
print(value)
```

가변 인수를 받는 형식 인수

문자열 전달

초기값을 갖는 매개변수

함수

- 전역변수 값을 **swap** 하는 함수 구현

```
a = 10
b = 20
def swap_func():
        global a, b
        a, b = b , a

print("a : %d, b : %d" %(a,b))
swap_func()
print("a : %d, b : %d" %(a,b))
```

- 문자열 데이터 임의의 문자열로 변환해 암호화 구현

```
def encrypt(msg):
        for ch in msg:
                if ch in encbook:
                        msg = msg.replace(ch, encbook[ch])
        return msg

encbook = { 'p':'%', 'y':'(', 't':'#', 'h':'=', 'o':'@', 'n':'!' }

res_msg = encrypt("I love python programming")
print(res_msg)
```

사전에 있는 **key** 와 일치하는 **value** 값으로 변환

```
I l@ve %(#=@! %r@grammi!g
[Finished in 0.3s]
```

함수

- 알파벳 대문자**(A~Z)** 까지를 **(0~25)**로 매칭 시킨 후 특정 문자에 대한 매칭 값 반환 함수 구현

```
def MakeAlphaValue(key):
        mylist = [ (chr(x+65), x) for x in range(26) ]

        mydic =  {}
        for dt in mylist:
                alpha, index = dt[0], dt[1]
                mydic[alpha] = index

        print(mylist,"\n")
        print(mydic, "\n")

        if key in mydic:
                k = mydic[key]
        else:
                return None

        return k


key_data = MakeAlphaValue('L')
print("key_data :", key_data)
```

```
mydic = dict(mylist)
```

```
[('A', 0), ('B', 1), ('C', 2), ('D', 3), ('E', 4), ('F', 5),
('G', 6), ('H', 7), ('I', 8), ('J', 9), ('K', 10), ('L', 11),
('M', 12), ('N', 13), ('O', 14), ('P', 15), ('Q', 16), ('R',
17), ('S', 18), ('T', 19), ('U', 20), ('V', 21), ('W', 22),
('X', 23), ('Y', 24), ('Z', 25)]

{'A': 0, 'B': 1, 'C': 2, 'D': 3, 'E': 4, 'F': 5, 'G': 6, 'H': 7,
'I': 8, 'J': 9, 'K': 10, 'L': 11, 'M': 12, 'N': 13, 'O': 14,
'P': 15, 'Q': 16, 'R': 17, 'S': 18, 'T': 19, 'U': 20, 'V':
21, 'W': 22, 'X': 23, 'Y': 24, 'Z': 25}

key_data : 11
[Finished in 0.2s]
```

클래스

- 클래스 생성

```
class myclassTest():
        var = 78                                    # 모든 인스턴스가 공유하는 클래스 속성 변수
        def __init__(self, data):
                var = 20                            # __init__ 메서드 내부 지역변수
                self.var = data                     # 인스턴스 속성 변수

myinstance1 = myclassTest(30)
print("instant1 : ", myinstance1.var)
print(myclassTest.var)

myinstance2 = myclassTest(50)
print("instant2 : ", myinstance2.var)
print(myclassTest.var)
```

클래스 인스턴스가 생성된 직후 자동으로 호출되는 메소드
(객체 속성 변수 초기화 역할)

클래스 이름공간 내의 클래스 속성 변수 접근

인스턴스 이름공간 내의 객체 속성 변수 접근

```
instant1 :  30
78
instant2 :  50
78
[Finished in 0.2s]
```

클래스

- 클래스 메소드 장식자

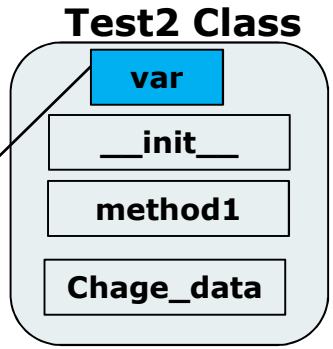
```
class Test2():
        var = 78
        def __init__(self, data):
                self.var = data
                print("_init_ self.var :", self.var)

        def method1(self):
                print("method1 Test2.var : ", Test2.var)
                print("method1 self.var : ", self.var)

        @classmethod
        def chage_data(cls,de):
                cls.var = de
```

클래스 속성 변수

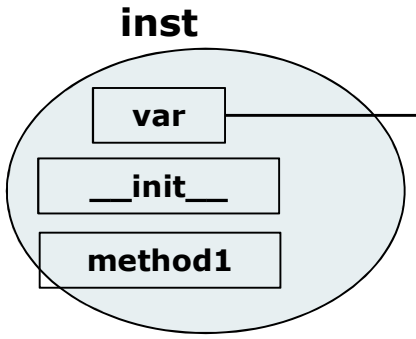
인스턴스 속성 변수

```
inst = Test2(50)
print("inst var : ",inst.var)

inst3 = Test2(60)
print("inst3 var : ",inst3.var)

inst.method1()
inst3.method1()

Test2.chage_data(217)
print("Test2.var : ",Test2.var)
Test2.var = 33
print("Test2.var : ", Test2.var)
```

클래스 메소드 장식자 활용
클래스 속성변수 접근



인스턴스들이 공유하는
클래스 속성 변수



self를 통해 생성된 개별 인스턴스 속성 변수

* 주의 : 클래스 속성 변수와 인스턴스 속성 변수 이름 일

클래스

- 클래스 내부 스코핑룰

```
class HouseClass():
        Company = "python Academy"
        def __init__(self, year, address, price):
                self.year = year
                self.address = address
                self.price = price
        def show_company(self):
                print(self.Company)
        def Change_company(self,name):
                self.Company = name
        def show_info(self):
                print("""This house was built by {} in {}, address : {}, price : {} """
                        .format(self.Company, self.year, self.address, self.price))

houseA = HouseClass(2019, "Guro", 34.56)
houseA.show_company()
houseA.Change_company("MDS Academy")
houseA.show_company()
houseA.show_info()
houseB = HouseClass(2020, "pangyo", 999.99)
houseB.show_info()
```

인스턴스 속성 변수가 없다면 스코핑 룰에 따라 클래스 속성 변수를 찾음

인스턴스 속성변수를 생성해서 전달인자 값 저장

클래스

- 클래스 속성 변수 공유해서 활용

```
class HouseClass():
        Company = "python Factory"
        def __init__(self, year, address, price):
                self.year = year
                self.address = address
                self.price = price
        @classmethod
        def show_company(cls):
                print(cls.Company)
        @classmethod
        def Change_company(cls,name):
                cls.Company = name
        def show_info(self):
                print("""This house was built by {} in {}, address : {}, price : {} """
                        .format(HouseClass.Company, self.year, self.address, self.price))

houseA = HouseClass(2019, "Guro", 34.56)
houseA.show_company()
houseA.Change_company("MDS Academy")
houseA.show_company()

houseA.show_info()

houseB = HouseClass(2020, "pangyo", 999.99)
HouseClass.Company = "Hancommds"
houseB.show_info()
```

클래스 메소드 정의 : 클래스 속성 변수 접근

클래스 메소드 정의 : 클래스 속성 변수 수정

모든 인스턴스는 클래스 속성 변수 **Company** 공유해서 사용

클래스

- 클래스 구현 예제 **(Calculator 클래스)**

```
class Calculator():
        def __init__(self, mylistdata):
                self.mylist = mylistdata
        def sum(self):
                self.total = 0
                for x in self.mylist:
                        self.total += x
                return self.total
        def avg(self):
                list_len = len(self.mylist)
                self.avg = self.total / list_len
                return self.avg

cal1 = Calculator([1,2,3,4,5])
print(cal1.sum())
print(cal1.avg())

cal2 = Calculator([6,7,8,9,10])
print(cal2.sum())
print(cal2.avg())
```

클래스

- 클래스 상속

```
class Accout():
        def __init__(self, money):
                self.balance = money
        def deposit(self, money):
                self.balance += money
        def withdraw(self, money):
                self.balance -= money
        def show_Accout(self):
                print("balance : {} 원".format(self.balance))
```

상속

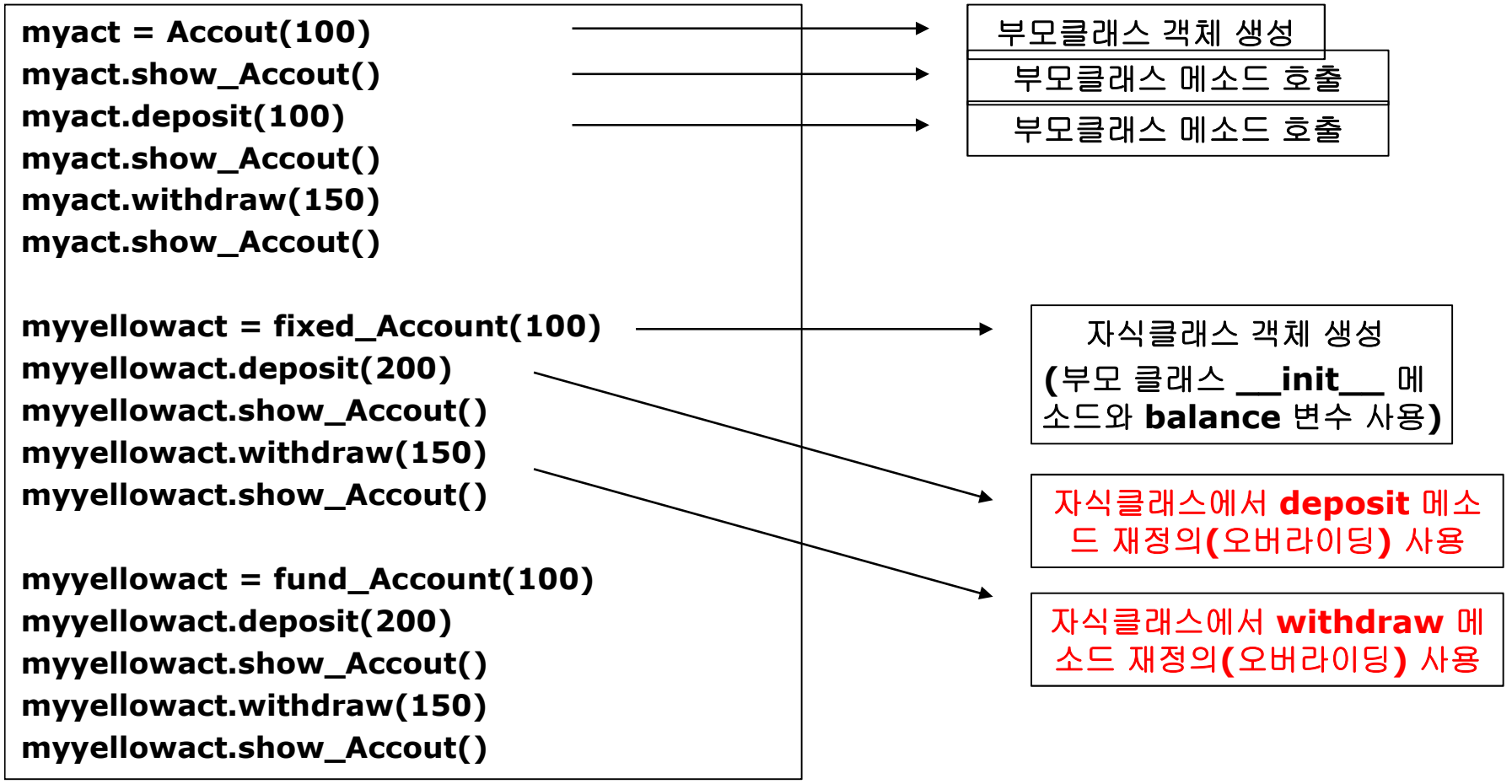
상속

```
class fixed_Account(Accout):
        def deposit(self, money):
                self.balance += money*1.07
        def withdraw(self, money):
                self.balance -= money + 10
```

```
class fund_Account(Accout):
        def deposit(self, money):
                self.balance += money*2.17
        def withdraw(self, money):
                self.balance -= money + 150
```

클래스

- 클래스 상속


myact = Accout(100)
myact.show_Accout()
myact.deposit(100)
myact.show_Accout()
myact.withdraw(150)
myact.show_Accout()
myyellowact = fixed_Account(100)
myyellowact.deposit(200)
myyellowact.show_Accout()
myyellowact.withdraw(150)
myyellowact.show_Accout()
myyellowact = fund_Account(100)
myyellowact.deposit(200)
myyellowact.show_Accout()
myyellowact.withdraw(150)
myyellowact.show_Accout()
부모클래스 객체 생성
부모클래스 메소드 호출
부모클래스 메소드 호출
자식클래스 객체 생성
(부모 클래스 __init__ 메소드와 balance 변수 사용)
자식클래스에서 deposit 메소드 재정의(오버라이딩) 사용
자식클래스에서 withdraw 메소드 재정의(오버라이딩) 사용

클래스

HANCOM 한컴MDS 아카데미

- 클래스 상속 (오버라이딩 – 새로운 속성 추가)

```
class Account():
        def __init__(self, money):
                self.balance = money
        def deposit(self, money):
                self.balance += money
        def withdraw(self, money):
                self.balance -= money
        def show_Account(self):
                print("balance : {} 원
".format(self.balance))
```

```
myact = stock_accout("MDS", 500)
myact.deposit(200)
myact.show_Account()
```

부모로 부터 물려받은 **balance** 변수 초기화하기 위해 부모 클래스 **__init__**함수 호출

만약 자식클래스에서 **balance**변수 생성하면 자식클래스 이름공간의 **balance**변수가 되어 버림

```
class stock_accout(Account):
        def __init__(self,name, money):
                Account.__init__(self, money)
                self.name = name
        def deposit(self, money):
                self.balance += money * 1.37
        def withdraw(self, money):
                self.balance -= money + 50
        def show_Account(self):
                print("Account owner : {}".format(self.name))
                Account.show_Account(self)     # print("balance : {} 원
".format(self.balance)) 와 동일
```

오버라이딩된 **__init__**메소드에서 새로운 속성 추가 기능

클래스

- 클래스 상속 **(super()함수 : 함수가 반환하는 객체를 통해서 부모클래스 속성 접근)**

```
class Account():
        def __init__(self, money):
                self.balance = money
        def deposit(self, money):
                self.balance += money
        def withdraw(self, money):
                self.balance -= money
        def show_Account(self):
                print("balance : {} 원
".format(self.balance))
```

```
class stock_accout(Account):
        def __init__(self,name, money):
                super().__init__(money)
                self.name = name
        def deposit(self, money):
                self.balance += money * 1.37
        def withdraw(self, money):
                self.balance -= money + 50
        def show_Account(self):
                print("Account owner : {}".format(self.name))
                super().show_Account()    # print("balance : {} 원
".format(self.balance)) 와 동일
```

클래스

- 클래스 연산자 오버로딩 (내장된 연산자만 가능)

```
class mylistclass():
        def __init__(self, data):
                self.mylistdata = data
        def show_list(self):
                print(self.mylistdata)
        def __sub__(self, other):
                myset1 = set(self.mylistdata)
                myset2 = set(other.mylistdata)
                myres = myset1 - myset2
                return list(myres)
```

객체(-) 기능 오버로딩

```
mydata1 = [ 1, 2, 3, 4, 5]
mydata2 = [ 7, 9, 4, 2, 1]
myinst1 = mylistclass(mydata1)
myinst2 = mylistclass(mydata2)
myinst1.show_list()
myinst2.show_list()

result = myinst1 - myinst2
print("myinst1 - myinst2 :", result)
```

```
[1, 2, 3, 4, 5]
[7, 9, 4, 2, 1]
myinst1 - myinst2 : [3, 5]
[Finished in 0.2s]
```

클래스

- **list** 클래스 상속 및 연산자 오버로딩

```
class mylistclass(list):
        def __init__(self, name):
                self.name = name
        def __add__(self, other):
                mysumlist = []
                mysumlist.extend(self)
                mysumlist.extend(other)
                return mysumlist
```

객체 **(+)** 기능 오버로딩

```
myinst1 = mylistclass("myinst1 data :")
for x in range(1,5):
        myinst1.append(x)
print(myinst1.name, myinst1)

myinst2 = mylistclass("myinst2 data :")
for x in range(5, 10):
        myinst2.append(x)
print(myinst2.name, myinst2)

result = myinst1 + myinst2
print("result :", result)
```

myinst1은 **list** 클래스 상속 객체 임으로 **list** 클래스 메서드 사용

```
myinst1 data : [1, 2, 3, 4]
myinst2 data : [5, 6, 7, 8, 9]
result : [1, 2, 3, 4, 5, 6, 7, 8, 9]
[Finished in 0.2s]
```

클래스

- **list** 클래스 상속 및 연산자 오버로딩

```
class mylistclass(list):
        def __init__(self, name):
                self.name = name
        def __sub__(self, other):
                myset1 = set(self)
                myset2 = set(other)
                myres = myset1 - myset2
                return list(myres)
```

객체 **(-)** 기능 오버로딩

```
myinst1 = mylistclass("myinst1 data :")
for x in range(1,8):
        myinst1.append(x)
print(myinst1.name, myinst1)

myinst2 = mylistclass("myinst2 data :")
for x in range(3, 10):
        myinst2.append(x)
print(myinst2.name, myinst2)

result = myinst1 - myinst2
print("myinst1 - myinst2 :", result)
```

myinst1은 **list** 클래스 상속 객체 임으로 **list** 클래스 메서드 사용

```
myinst1 data : [1, 2, 3, 4, 5, 6, 7]
myinst2 data : [3, 4, 5, 6, 7, 8, 9]
myinst1 - myinst2 : [1, 2]
[Finished in 0.2s]
```

Thank you

㈜한컴MDS www.hancommds.com

본사 13493 경기도 성남시 분당구 대왕판교로 644번길 49 한컴타워 3,4층 031-627-3000
연구소 13487 경기도 성남시 분당구 판교로 228번길 17 판교세븐벤처밸리 2단지 1동 9층 031-600-5000