KU KONKUK UNIVERSITY

클래스

자연어처리연구실

클래스 (Class)

- 클래스
  - 자료와 그 자료에 대한 기능(함수)를 묶어 놓은 프로그래밍 단위
    - 속성(attribute), 클래스 변수(class variable): 클래스에 정의된 자료
    - 메소드(method): 클래스에 정의된 함수
  - 클래스를 메모리에 할당하여 사용하면 인스턴스(객체)가 됨
    - 클래스는 추상적인 개념이며 인스턴스는 그것을 구체화하여 실현한 것
  - 예제
    - 사람 클래스: { 눈, 코, 입, 눈을 감다, 숨을 쉬다, … }
    - 자동차 클래스: { 핸들, 기어, 엑셀, 방향을 틀다, 가속하다, … }
    - 사람 인스턴스: 홍길동, 임꺽정
    - 자동차 인스턴스: 내 차, 친구 차, …

클래스 (Class)

- 클래스 vs. 인스턴스
  - 클래스: 설계도
  - 인스턴스: 설계도에 따라 구현된 것
  - 예제
    - 클래스: 붕어빵틀
    - 인스턴스: 붕어빵1, 붕어빵2, …
  - 형식

클래스

```
class 클래스명:
    변수 정의, …
    함수 정의, …
```

인스턴스

```
변수 = 클래스명()
```

클래스 (Class)

- 'class'는 클래스를 만들 때 쓰이는 예약어
- 'class' 뒤에는 바로 클래스명을 써주어야 함
- 클래스명 뒤에 상속(추후 설명)할 클래스가 있으면 상속할 클래스명을 씀
- 클래스 내부에는 속성과 메소드를 정의함

```
class 클래스명(상속 클래스명):
    속성1
    속성2
    …
    def 메소드1(self, 인자1, 인자2,…):
        <수행할 문장 1>
        <수행할 문장 2>
        …
    def 메소드2(self, 인자1, 인자2,…):
        <수행할 문장1>
        <수행할 문장2>
        …
    …
```

클래스 (Class)

Calc_module.py

```
class Calc:

    my_name = "I'm Calc Module!"

    def add(self, a, b):
        result = a + b
        print("{0:f} + {1:f} = {2:f}".format(a,b,result))
    def sub(self, a, b):
        result = a - b
        print("{0:f} - {1:f} = {2:f}".format(a,b,result))
    def mul(self, a, b):
        result = a * b
        print("{0:f} * {1:f} = {2:f}".format(a,b,result))
    def div(self, a, b):
        if b != 0:
            result = a / b
            print("{0:f} / {1:f} = {2:f}".format(a,b,result))
        else:
            print("Divided by zero!")
```

인스턴스화 될 때 각 인스턴스자신을 다른 것과 구분하기 위한 디폴트 인자!!

Test_Calc.py

```
#from Calc_module import *
from Calc_module import Calc

num=Calc()
print(num.my_name)
num.add(1,2)
num.sub(1,2)
num.mul(1,2)
num.div(1,0)
```

self는 명시적으로 호출하지 않음

```
I'm Calc Module!
1.000000 + 2.000000 = 3.000000
1.000000 - 2.000000 = -1.000000
1.000000 * 2.000000 = 2.000000
Divided by zero!
```

Edited by Harksoo Kim

클래스 연산자 메소드

함수	설명	예제
__init__	생성자(Constructor): 인스턴스가 만들어 질 때 호출	
__del__	소멸자(Destructor): 인스턴스가 사라질 때 호출	
__add__	연산자 "+"	x + y
__or__	연산자 "|"	x | y
__repr__	print	print(x)
__call__	함수호출 X()했을 때 호출	
__getattr__	자격부여	x.method
__getitem__	인덱싱	x[i]
__setitem__	인덱스 치환	x[key] = value
__getslice__	슬라이싱	x[i:j]
__cmp__	비교	x > y

Edited by Harksoo Kim

생성자 (__init__)

- 생성자
  - 클래스가 인스턴스화될 때 자동 수행되는 메소드
  - 주로 속성 값 초기화가 필요할 때 재정의

```
class Calc:

    my_name = "I'm Calc Module!"
    scale = 0 # 생성자에서 정의할 경우에 이곳에 선언해도 되고 안해도 됨

    def __init__(self,scale=1):
        self.scale = scale

    def add(self, a, b):
        result = (a + b)*self.scale
        print("({0:f} + {1:f}) * {2:f} = {3:f}".format(a,b,self.scale,result))
```

```
num1=Calc()
num1.add(1,2)
num2=Calc(10)
num2.add(1,2)
```

```
(1.000000 + 2.000000) * 1.000000 = 3.000000
(1.000000 + 2.000000) * 10.000000 = 30.000000
```

Edited by Harksoo Kim

소멸자 (__del__)

- 소멸자
  - 인스턴스가 소멸(메모리에서 제거)될 때 자동 수행되는 메소드
  - 주로 메모리에서 제거할 것이 있을 때 재정의

```
class Calc:

    my_name = "I'm Calc Module!"
    scale = 0 # 생성자에서 정의할 경우에 이곳에 선언해도 되고 안해도 됨

    def __init__(self,scale=1):
        self.scale = scale

    def __del__(self):
        self.scale =1
        print("Scale is reset to 1.")

    def add(self, a, b):
        result = (a + b)*self.scale
        print("({0:f} + {1:f}) * {2:f} = {3:f}".format(a,b,self.scale,result))
```

```
num1=Calc()
num1.add(1,2)
num2=Calc(10)
num2.add(1,2)
del(num2)
```

```
(1.000000 + 2.000000) * 1.000000 = 3.000000
(1.000000 + 2.000000) * 10.000000 = 30.000000
Scale is reset to 1.
```

Edited by Harksoo Kim

상속 (inheritance)

- 상속
  - 기존 클래스의 일부를 수정하여 사용하는 메커니즘
  - 상속을 해주는 클래스를 부모 클래스(parent class), 상속을 받는 클래스를 자식 클래스(child class)라고 함
  - 자식 클래스는 부모의 속성과 메소드를 사용할 수 있으며, 기능을 개선하여 수정할 수 있음
    - 실세계 예: 부모의 자동차를 상속 받아 타이어(속성)를 바꾸고 하늘 날기(메소드)을 추가하는 것
  - 형식

```
class 클래스명(상속 클래스명):
    속성
    ...
    def 메소드(self, 인자1, 인자2,...):
        <수행할 문장 1>
        <수행할 문장 2>
        ...
```


상속 (inheritance)

```
class ParentCalc:
    def add(self, a, b):
        result = a + b
        print("{0:f} + {1:f} = {2:f}".format(a,b,result))
    def sub(self, a, b):
        result = a - b
        print("{0:f} - {1:f} = {2:f}".format(a,b,result))

class ChildCalc(ParentCalc):
    def add(self, *a):
        s = "{0:f}".format(a[0])
        for i in a[1:]:
            s = s + " + " + "{0:f}".format(i)
        result = sum(a)
        print("{0:s} = {1:f}".format(s,result))

num=ChildCalc()
num.add(1,2,3,4)
num.sub(1,2)
```

함수 오버로딩
(function overloading)

```
1.000000 + 2.000000 + 3.000000 + 4.000000 = 10.000000
1.000000 - 2.000000 = -1.000000
```


접근 권한 (access modifier)

- 접근 권한
  - Public (디폴트)
    - 자식에게 모두 상속되며 자식이 아닌 경우도 접근 가능
  - Protected (접근 제한하려면 복잡함. 추천 안함)
    - 자식에게 상속되며 자식만 접근 가능
    - 형식: 속성명이나 메소드명에 '_'를 붙임
  - Private
    - 자식에게도 상속되지 않으며 본인만 접근 가능
    - 형식: 속성명이나 메소드명에 '__'를 붙임

접근 권한 (access modifier)

```
class ParentCalc:
    my_name = "Public ParentCalc"
    __my_name = "Private ParentCalc"
    def show_name(self):
        print(self.my_name)
        print(self.__my_name)

class ChildCalc(ParentCalc):
    my_name = "Public ChildCalc"
    __my_name = "Private ChildCalc"
    def show_name(self):
        print(super().my_name)
        print(self.my_name)
        print(super().__my_name)

num1=ParentCalc()
num1.show_name()
print("")
print(num1.my_name)
print(num1.__my_name)
```

```
Public ParentCalc
Private ParentCalc

Public ParentCalc
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-92-ce6c303f3432> in <module>
     18 print("")
     19 print(num1.my_name)
---> 20 print(num1.__my_name)
```

```
num1=ParentCalc()
num1.show_name()
print("")
num2=ChildCalc()
num2.show_name()
```

```
Public ParentCalc
Private ParentCalc

Public ParentCalc
Public ChildCalc

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-94-d94fb3964ce8> in <module>
     18 print("")
     19 num2=ChildCalc()
---> 20 num2.show_name()

<ipython-input-94-d94fb3964ce8> in show_name(self)
     12         print(super().my_name)
     13         print(self.my_name)
---> 14         print(super().__my_name)
```

연산자 오버라이딩 (operator overriding)

- 연산자 오버라이딩
  - 인스턴스 간에 연산을 수행할 수 있도록 연산자(+, -, *, /, ...)를 재정의하여 사용하는 것
  - 클래스 연산자 메소를 재정의
    - __add__(self, other): self 인스턴스 + other 인스턴스
    - __sub__(self, other) : self 인스턴스 - other 인스턴스
    - __mul__(self, other) : self 인스턴스 * other 인스턴스
    - __truediv__(self, other) : self 인스턴스 / other 인스턴스
    - 기타 다양한 연산자가 존재

연산자 오버라이딩 (operator overriding)

```
class Point:
    def __init__(self,x,y):
        self.x=x
        self.y=y
    def __add__(self,other):
        self.x += other.x
        self.y += other.y
        return self
    def show(self):
        print("({0:d},{1:d})".format(self.x,self.y))

a = Point(1,2)
a.show()

b = Point(3,4)
b.show()

a = a+b
a.show()
```

```
(1,2)
(3,4)
(4,6)
```

계산된 값을 다른 인스턴스에 할당(=)하기 위해서 자신을 리턴

예외처리 (exception handling)

- 예외처리
  - 프로그래밍 중 발생할 수 있는 여러 에러를 모아서 처리하는 메커니즘
  - 다양한 예외가 존재
    - 0으로 나누는 경우
    - 파일이 없는데 open하는 경우
    - 리스트의 범위를 초과하여 인덱싱하는 경우

```
>>> 4/0

Traceback (most recent call last):
  File "<pyshell#40>", line 1, in <module>
    4/0
ZeroDivisionError: integer division or modulo by zero
>>> f = open("not exist file", 'r')

Traceback (most recent call last):
  File "<pyshell#39>", line 1, in <module>
    f = open("not exist file", 'r')
IOError: [Errno 2] No such file or directory: 'not exist file'
```

```
>>> a = [1,2,3]
>>> a[4]

Traceback (most recent call last):
  File "<pyshell#42>", line 1, in <module>
    a[4]
IndexError: list index out of range
```


예외처리 (exception handling)

```
try:
    ...
except [발생에러[as 에러메시지변수]]:
    ...
finally:
    ...
```

[] 부분과 finally 이하는 생략 가능

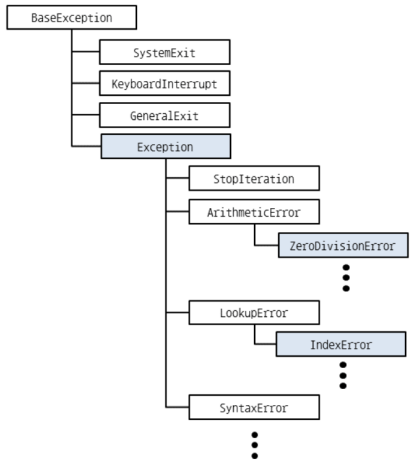
에러가 발생하면 except문을 실행

에러 발생 시 except문에 미리 정해놓은 에러와 일치할 때만 except문 실행

에러 발생 여부와 상관없이 무조건 실행

예외처리 (exception handling)

- 예외 계층도



예외처리 (exception handling)

```
num1 = int(input("num1="))
num2 = int(input("num2="))
try:
    num3 = num1/num2
    print("{0:d} / {1:d} = {2:f}".format(num1,num2,num3))
except ZeroDivisionError as e:
    print(e)
finally:
    print("The end!")
```

```
num1=4
num2=0
division by zero
The end!
```

```
f = open("test.txt","r")
try:
    ...
except IOError as e:
    print(e)
finally:
    f.close()
```

어떤 경우에도 파일은 닫아야 하기 때문

실습

- 2차원 공간에서 점을 나타내는 Point 클래스를 작성하고, 그것을 상속 받아 Box 클래스를 작성하시오.
  - Point Class
    - 속성
      - 좌표 (x, y)
    - 메소드
      - move(dx, dy): 좌표 (x, y)를 (dx, dy) 만큼 이동
      - show(): 좌표 (x, y)를 출력
  - Box Class
    - 속성
      - 좌상단, 우하단 포인트 좌표 (p1, p2)
    - 메소드
      - move(dx, dy): 좌상단, 우하단 포인트 좌표를 (dx, dy) 만큼 이동
      - Show(): 좌상단, 우하단 포인트 좌표 출력

실습

?

질의응답

Q&A

Homepage: http://nlp.konkuk.ac.kr
E-mail: nlpdrkim@konkuk.ac.kr

Edited by Harksoo Kim