SSAFY 파이썬 기초 - 10

TIL 2019.01.15 09:36

위젯을 만들려면 그래픽 인터페이스를 제어할 수 있어야한다.

=> tkinter

엄청 간단한 그래픽 유저 인터페이스이다. 윈도우 같은 프로그램을 아주 간단한 코드로 만들어준다.

command + R 해도 자주 쓰는 걸 올려놓고, 클릭만 해서 무언가 되게 만들기

```
cd scripts/
code .
```

scripts에서 에디터 열고 시작하고, widget.py를 하나 만든다.

실전 프로그래밍에서는 * 쓰면 안 되지만 학습하는 구간이니까 쓸 것이다. c++ 도 std:: 이런 네임 스페이스를 빼면 나중에 어디서 나왔는지 모르니까 그렇게 짜면 안 된다.

from import 뒤에 * 을 붙이면 파이썬이 이런 식으로 구현해줬다고 생각하면 된다. (알아서 줄여주는 것)

```
# widget.py
```

```
TK = tkinter.Tk
Label = tkinter.Label
```

바명룡

이런 식으로 메모리 공간 안에 다 올려놓는다. 되게 비효율적이다...

머신러닝 (11)

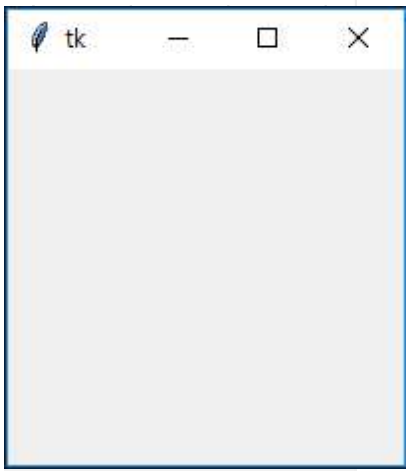
다음과 같이 쓰면 위젯이 만들어진다.

자료구조&알고리즘 (12)

```
# widget.py
from tkinter import *
# import tkinter 쓰면 아래에서 부를 때마다 tkinter. 쓰기 귀찮으니까!

root = Tk()

root.mainloop()
# 사람들의 클릭이 있을 때, 그걸 계속 확인하고 실행해주는 조그만 위젯
```

머신러 머신러 알 윈

결과물



Label()은 대문자니까 클래스라는 것을 알 수 있다.

리신 발동!

```
# tkinter.py
class Label:
    def __init__(self, program, text=""):
```

이런 식으로 작성되었을 거라고 생각할 수 있다.

```
# widget.py
# import tkinter 쓰면 아래에서 부를 때마다 tkinter. 쓰기 귀찮으니까!
from tkinter import *

# root라는 윈도우를 만들었다.
root = Tk()
```

```
# Label(어떤 tkinter 윈도우/프로그램에 넣을지, text = "")
# label = Label이라는 클래스를 이니셜라이즈!
label = Label(root, text = "Hello")

# 좌표값 지정 (.pack()이라는 메소드, 가장 위에서부터 쌓아나가준다.)
label.pack()

# 사람들의 클릭이 있을 때, 그걸 계속 확인하고 실행해주는 조그만 위젯
root.mainloop()
```


버튼도 만들 수 있다.

#widget.py

```
# import tkinter 쓰면 아래에서 부를 때마다 tkinter. 쓰기 귀찮으니까!
from tkinter import *

# root라는 윈도우를 만들었다.
root = Tk()

# Label(어떤 tkinter 윈도우/프로그램에 넣을지, text = "")
# label = Label이라는 클래스를 이니셜라이즈!
label = Label(root, text = "Hello")
label2 = Label(root, text = "John Kang's widget")

# 버튼 만들기 (라벨 만드는 과정과 비슷하다) 1. 이니셜라이즈, 2. pack
btn = Button(root, text="This is a button")
btn.pack()

# 좌표값 지정 (.pack()이라는 메소드, 가장 위에서부터 쌓아나가준다.)
label.pack()
label2.pack()

# 사람들의 클릭이 있을 때, 그걸 계속 확인하고 실행해주는 조그만 위젯
root.mainloop()
```

지금 버튼 눌러도 아무 일도 안 일어난다.

홈

버튼이 아래로 내려가게 하고 싶으면 pack의 순서만 바꿔주면 된다.

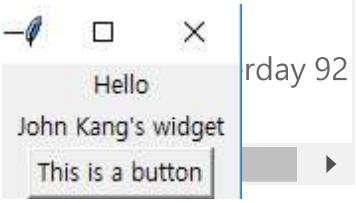
방명록

widget.py

```
# import tkinter 쓰면 아래에서 부를 때마다 tkinter. 쓰기 귀찮으니까!
from tkinter import *

# root라는 윈도우를 만들었다.
root = Tk()

# Label(어떤 tkinter 윈도우/프로그램에 넣을지, text = "")
# label = Label이라는 클래스를 이니셜라이즈!
label = Label(root, text = "Hello")
label2 = Label(root, text = "John Kang's widget")

# 버튼 만들기 (라벨 만드는 과정과 비슷하다) 1. 이니셜라이즈, 2. pack
btn = Button(root, text="This is a button")

# 좌표값 지정 (.pack()이라는 메소드, 가장 위에서부터 쌓아나가준다.)
label.pack()
label2.pack()
btn.pack()

# 사람들의 클릭이 있을 때, 그걸 계속 확인하고 실행해주는 조그만 위젯
root.mainloop()
```

ABOUT ME

실행결



이게 바로 OOP. 다른 사람이 만들어 놓은 걸 가져와서 어떻게 작동하는지 유추 가능하다!

+ 색도 변화 가능하다

widget.py

```
# import tkinter 쓰면 아래에서 부를 때마다 tkinter. 쓰기 귀찮으니까!
from tkinter import *
# 버튼 누르면 brower 켜지게 할 거임
import webbrowser

# 이 함수를 button의 세 번째 인자로 넣어줄 수 있다.
def browser():
    webbrowser.open("https://www.daum.net")

# root라는 윈도우를 만들었다.
```

```
root = Tk()

# Label(어떤 tkinter 윈도우/프로그램에 넣을지, text = "")
# label = Label이라는 클래스를 이니셜라이즈!
label = Label(root, text = "Hello", fg="red", bg="blue")
label2 = Label(root, text = "John Kang's widget")

# 버튼 만들기 (라벨 만드는 과정과 비슷하다)
# command에 버튼 눌렸을 때 작동할 함수 이름을 그대로 써준다.
btn = Button(root, text="This is a button", command = browser)

# 좌표값 지정 (.pack()이라는 메소드, 가장 위에서부터 쌓아나가준다.)
label.pack()
label2.pack()
btn.pack()

# 사람들의 클릭이 있을 때, 그걸 계속 확인하고 실행해주는 조그만 위젯
root.mainloop()
```

결과물

ABOUT ME

위에 command에 browser를 넣었는데, ()를 쓰지 않았다. 이전에 map함수를 쓸 때도 map(int, 대상) 이런 식으로 ()를 쓰지 않고 사용했었다.

```
def ahnyung():
    print("ahnyoung")
def ohiyou():
    print("ohiyou")
def hello():
    rprint("hello")
def greeter(func):
    func()
```

greeter(ohiyou) 이런 식으로 쓸 수 있다. 함수를 인자로 넣어서 쓸 수도 있다.
cf) 하이오더 펑션, 일급 객체로서의 함수라고 부른다.

cf) (파이썬의 리스트가 아님) 리스트는 다른 말로 연결 리스트 (linked list)라고 한다. 이걸 배우면 파이썬 버전 블록체인을 만들 수 있다. 노드가 연결이 되어있는 것인데, 그게 블록체인을 만드는 것과 다르지 않

다.
홈

< 선언과, instantiate, initialize 차이 >

```
# 클래스 선언
class Person():

# instantiate, 인스턴스를 만든다. ('='를 이용한 할당이 들어가 있다.)
a = Person()

# Initialize 여기에 값을 넣어서 초기화 하고 싶을 때
class Person:
    def __init__(self, name):
        self.name = name
a = Person("john")
```

=> 둘은 비슷한 개념이지만, Instantiate하면서 초기화를 안 할 수는 있다. 하지만 초기화를 하면 그게 포함된다.

< 함수 이름을 인자로 넣으면? >

```
def hello():
    return "hello"
```

주피터에서 hello() 를 호출 하면 hello가 나오는 것처럼 보인다. 그러면 출력이 아니라 ouput만 보인다.
저장하고 git bash창에서 python fn.py를 하면 실제로 결과값이 안 나온다.
print(hello()) 로 수정해야만 결과가 나온다.
print(hello) 를 그대로 출력하면 <function hello at 0x00000265A9B71F28> 함수 이름 + 함수 값
함수가 들어가있는 주소, 공간이다.

자바스크립트에서는

```
function hello(){
    return "hello"
}
```

hello() 호출하면 "hello"값이 나온다. hello만 치면 f라고 나온다. 함수값 그 자체를 이야기한다.

인기포스트

머신러 머신러 알 원
자바스크립트는
초 초 초 초
코 코 코 코
쨔 쨔 쨔 쨔
응 응 응 응

```
hi = function() {
    return "hi"
}
```

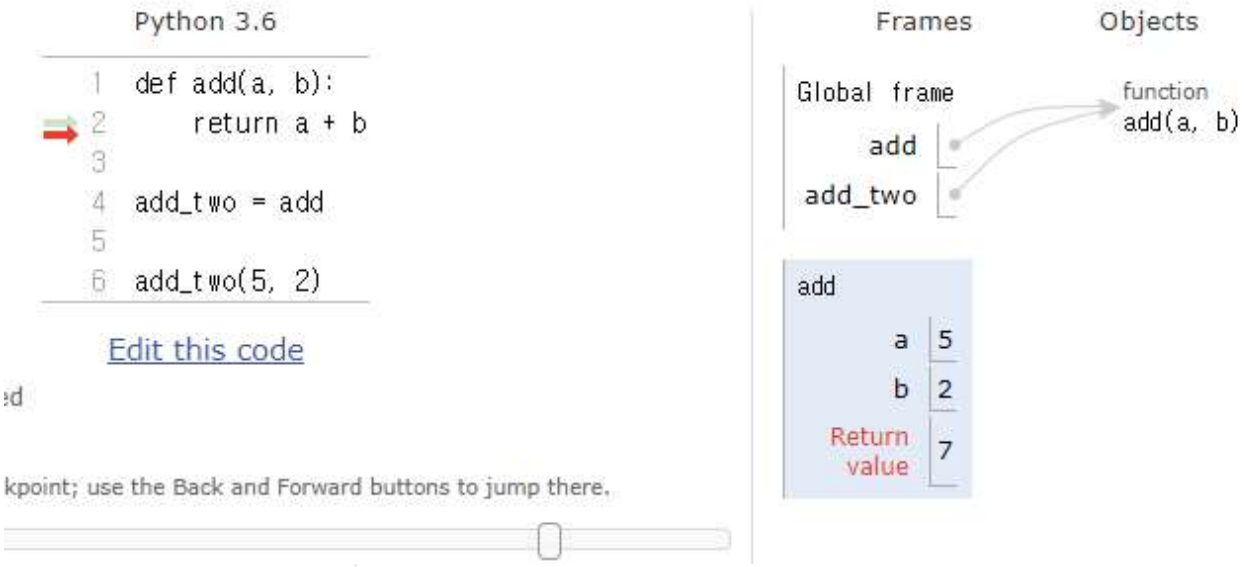
이렇게 기괴하게 쓴다. 변수에 함수를 담을 수 있다. 파이썬은 비슷하게 lambda로 비슷하게 할 수는 있지만, 위와 같은 건 지원하지 않는다.

ABOUT ME

Today 89 Yesterday 92
Total 4,037

```
def add(a, b):
    return a + b
add_two = add
add_two(5, 2)
```

파이썬 튜터로 위의 코드를 보면 add_two(5, 2)를 했지만 실제로는 add라는 함수가 돌아간다.



인기포스트

우리는 그동안 오른쪽에 있는 걸 왼쪽에 저장한다고 생각했었는데, 실제로는 오른쪽에 있는 값에 왼쪽의 이름이 바인딩 또는 할당된다는 말을 많이 한다.

파이썬은 일급 객체로서의 함수를 사용하는 언어. 일급 객체란? 함수조차 마치 다른 값들처럼 넘기고 넣고 할 수 있게 되었다. 그 기능을 지원하는게 바로 파이썬! 파이썬이 함수형 언어의 feature 도 가지고 있다.

ABOUT ME

1. 함수 값을 hello는 이름에 할당한 것
2. 함수를 호출(call)한다는 것은 hello () <- 호출 (연산자)

```
print(sum([1, 2, 3, 4]))
sum = "하하하 이제 너는 sum을 쓰지 못한다."
print(sum)
print(sum([1, 2, 3, 4]))
print = "print도 못 쓰지롱"
print()
```

sum에 원래 함수가 들어가 있었는데, 글자로 대체되었다.

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

add = 안에 function이 들어가 있다. 그래서 함수가 담긴 변수는 다 ()를 붙여서 실행이 가능한 것이다.

홈
방명록
머신러닝 (11)
자료구조&알고리즘 (12)
TIL (17)
안드로이드 스튜디오 (3)
인기포스트
머신러닝 머신러닝 알고리즘 원코드 코드 코드 코드
ABOUT ME
Today 89 Yesterday 92
Total 4,037

나: def라는 것은 정의된 함수 이름 = function() 이런 식으로 정의하는 것의 대체 문법이라고 생각된다. 그런데 파이썬에서는 def로 한 이름에 한 function만 들어가게 해 두었고, 이름2 = 정의된 함수 이름 이렇게 넣어도 원래 함수만 돌아가게 된다. 함수라는 클래스 안에 ()라는 메소드가 있는 걸까? -> 답변*

```
def greeter(f):
    f()
```

면 함수 안에 뭐가 들어오면 걔를 f에다가 담겠다. f = 이렇게. 그리고 출력은 없이 인자로 들어온 함수 f를 실행시킨다. 그래서 이 안에서 f가 들어온다는 것은 f = 는 hello()를 실행하는 것과 동일하다.

=> 파이썬에서 함수는 일급 객체이기 때문이다.

greeter는 return을 하지 않고, hello만 return을 한다면 결과가 None이 나온다. greeter가 혼자 물고 있기 때문이다. return값을 return 해줘야한다.

```
def hello():
    return "hello"

def greeter(f):
    return f()

greeter(hello)
```

*답변: 함수 클래스에서 튀어나온 객체처럼 저장이 되어있고, ()로 객체를 동작시킨다. 거대한 클래스의 인스턴스처럼 작동한다.

자바스크립트는

```
function hello(){
return "hello"
}

hello = function(){
return "hello"
}
```

자바스크립트에서는 본질적으로 똑같다. 파이썬에서도 동일하다. def 안에 들어간 모든 문장을 hello라는 변수에 담은 것과 똑같다.

파이썬에서 이걸 쓰려면

```
def hello():
    return "hello"
hello = lambda : "hello"
```

이렇게 써도 print(greeter(hi)) 가능하다.

< 일급 객체 함수 >

hi order function 를 직접 구현해볼 것이다.

```
# hof.py
```

```
list(map(int, ["1", "2"]))
```

맵은 iterable을 하나씩 돌면서 함수를 하나씩 적용한다. (mapping한다)

연습 - 내가 직접 map 함수와 filter함수 구현해보기!

```
# hof.py
# 기존의 일급 객체 함수 사용법
print(list(map(int, ["1", "2"])))

# 일급 객체 함수 직접 만들어보기
# my_map(함수, 리스트):
def my_map(func, input_list):
    # 0. 빈 리스트를 만들고
    # 1. 인자로 받은 리스트를 돌면서
    # 2. 인자로 받은 함수를 각각의 요소에 적용한 값을
    #    빈 리스트에 넣어서
    # 3. 빈 리스트를 리턴한다.
    new_list = []
    for item in input_list:
        new_list.append(func(item))
    return new_list

    # list comprehesion 버전
```

```
    # return [func(x) for x in input_list]

print(my_map(int, ["1", "2", "3"]))
print(my_map(str, [1, 2, 3]))

# filter 함수 사용법
def is_even(num):
    return num % 2 == 0
print(list(filter(is_even, [1, 2, 3, 4])))

# my_filter(참, 거짓을 리턴해주는 함수, 리스트)
def my_filter(func, input_list):
    empty_list = []
    for item in input_list:
        if func(item):
            empty_list.append(item)
    return empty_list

    # list comprehension 버전
    # return [x for x in input_list if func(x)]

print(my_filter(is_even, [1, 2, 3, 4, 5]))

# 람다 사용법
print(my_filter(lambda num: num % 2 ==0, [1, 2, 3, 4, 5]))
```


위에 있는 high order function 개념을 써서 먹일 수도 있지만, 람다를 써서도 먹일 수 있다.
람다는 그리스 기호로 function을 뜻하는 것과 똑같다. 그런데 프로그래밍에서는 익명 함수를 뜻한다.

< 람다 사용법 >

#lambda_test.py

```
# my_map 쓸거라서 print() 문은 주석처리해줘야한다.
import hof

# lambda랑 완전히 똑같은 예시 - def 사용
def add_two(num):
    return num + 2

# 2를 더하는 것이라 6이 나오고, 리스트의 모든 값에 2가 더해진다.
print(add_two(4))
print(hof.my_map(add_two, [1, 2, 3, 4]))

# 특정 함수를 넘겨줄 때 매번 정의하기가 귀찮다.
# 람다를 쓰면 다음 같이 입력, 출력으로 핵간단하게 만들 수 있다.
# print(hof.my_map(lambda 입력: 출력, [1, 2, 3, 4]))
print(hof.my_map(lambda num: num + 2, [1, 2, 3, 4]))
```

```
# 람다 함수를 집어넣을 수도 있다.
# ★코드 가독성을 흐리므로, 한 줄로 끝나지 않으면 람다 쓰지 마세요! - 파이썬 공식 추천
add_two = lambda num: num + 2
print(hof.my_map(add_two), [1, 2, 3, 4]))

# 연습
# 1. square 라는 변수에 lambda를 통해 제곱하는 함수를 할당
# 2. cube라는 변수에 세제곱
# 3. sqrt 변수에 제곱근 (math 활용)
square = lambda x: x ** 2
cube = lambda x: x ** 3
import math
sqrt = lambda x: math.sqrt(x)

print(hof.my_map(sqrt, [1, 2, 3, 4]))
```

인기포스트

클래스 - 인스턴스 간의 이름공간

어제 막내동생 비유를 통해 이름 공간을 설명했었다. 안에는 밖에를 볼 수 있지만, 우리는 동생 방에 들어가지 못하는 것을 생각해본다.

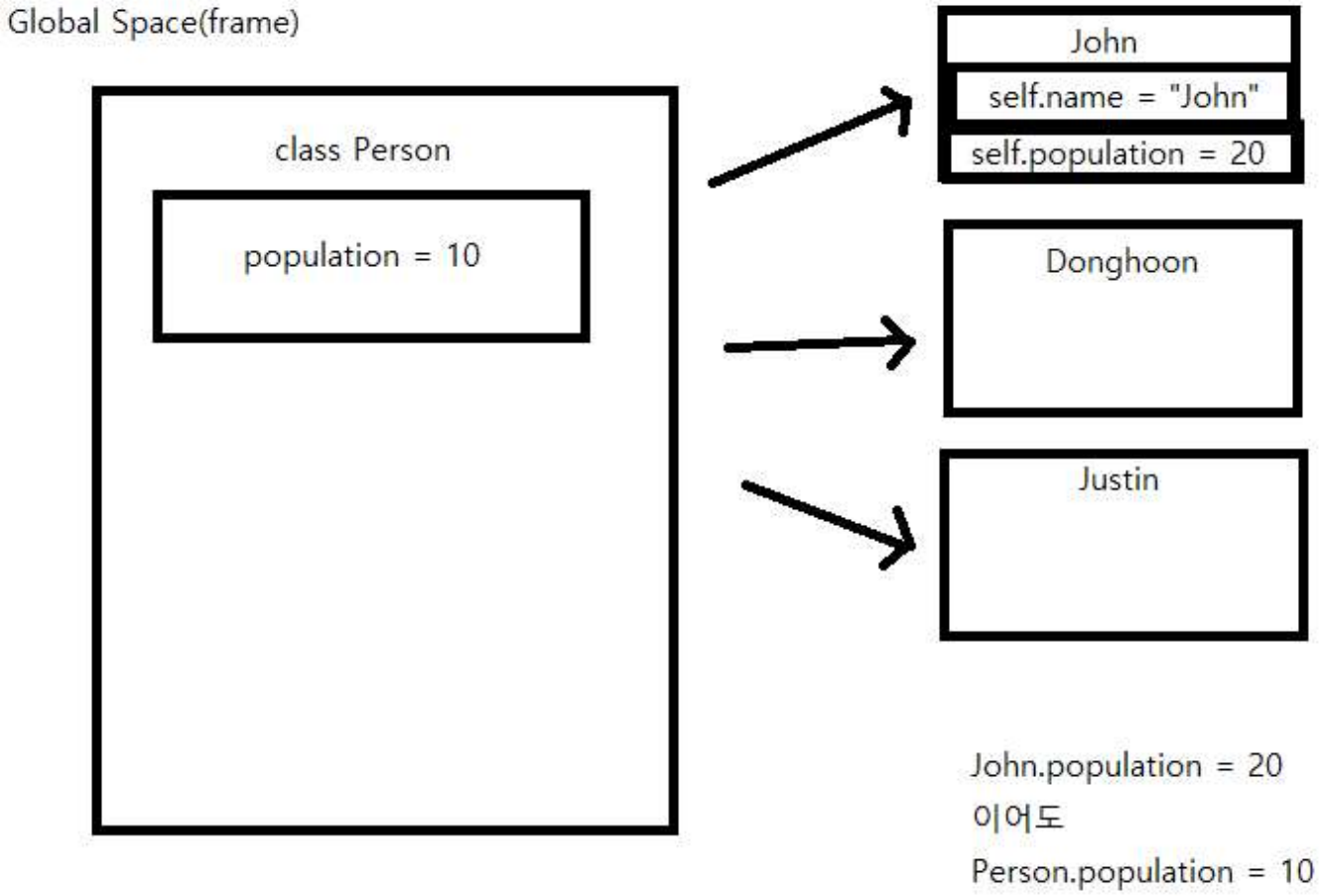
독특한게 클래스는 이름공간으로 접근 못하는 것처럼 보이지만, 타고 들어가면 접근 가능하다.

클래스를 만들어도 이름 공간이 생기고, 인스턴스를 만들어도 이름 공간이 생긴다. 인스턴스의 어트리뷰트는 인스턴스에 속한 친구들이다.

ABOUT ME

Today 89 Yesterday 92

Total 4,037



예시 코드

Today 89 Yesterday 92

Total 4,037

```
name = '?'

class Person:
    species = "인간"
    def __init__(self, name):
        self.name = name

    def greeting(self):
        print(f'{self.name}')
        print(f'{self.species}')
```

```
cr = Person('호날두')
cr.greeting()
```

해당하는 객체가 가지는 속성은 인스턴스 변수는 여기서 name 하나 뿐이다. 이 인스턴스가 가지는 고유한 데이터 공간이 name이라는 것 하나밖에 없다. .species는 클래스 공간에 있는데 왜 인스턴스 접근하듯이 쓸 수 있느냐? 인스턴스에 없는 경우에는 자연스럽게 클래스 공간에 있는지 찾기 때문이다.

cf) ORM 나오면 게시판 글 수 같은 거를 클래스에 저장한다. 클래스를 통해 해당하는 것을 접근하게끔 한다. 그래서 나중에 클래스 개념이 필요할 것이다.

아까 그냥은 읽어오는 친구이지만, cr.population = 20을 선언하는 순간 인스턴스에 population이 생긴다. self.population += 1은 **self.population** = **self.population** + 1을 축약한 것이다. 두 개가 더해져서 들어간다. 그런데 이 놈은 불리는 순간, self.population은 원래 데이터를 만드는 친구가 아니라 조회만 하는 친구이다. 자기한테 없으니까 인스턴스에 없으니 클래스로 간다. 클래스에서 10이라고 정의되어있으면 10+1이 인스턴스 변수 population에 들어가게 된다.

cf) 대입할 때는 항상 오른쪽부터 실행된다. 변수 = 오른쪽 먼저 실행!

나: 그러니까 정리해보면, 원래 공간 안에서 만들어지고 사라지는, 함수 안에서 사용되는 가변수들은 원래 밖에서 부를 수 없고 메모리에 잠깐 올라왔다가 사라지는데, 클래스나 인스턴스 안에서 사용하는 변수들은 사라지지 않는다. 이름을 붙여서 부르면 된다.

class 안의 def면은 {name} 나 아니면 밖에 이렇게만 찾고, {self.name}이면 인스턴스 안에서 찾고 없으면 클래스까지 간다.

```
def dongsang():
    def toilet():
        def doghouse():
            print(name)


class Person:
    Name = 'john'
    def __init__():
    def greeting():
```

def와 달리 class는 독특한 네임 공간이라 마치 모듈처럼 Person.name공간하면 된다. 접근 가능하다!

나: 인스턴스 안에는 변수만 들어가고, 나머지는 클래스로 올라가서 찾는 걸까? 그러면 나는 클래스의 주소만 가지고 있는 걸까?

홈

방명록

생성자 / 소멸자

머신러닝 (11)

자료구조&알고리즘 (12)

TIL (17)

```
#  생성시켜봅시다.
class Moosang:
    def __init__(self):
        print("저는 생성자입니다.")
    def __del__(self):
        print("저는 소멸자입니다.")
life = Moosang()
study = Moosang()
```

ABOUT ME

생성자입니다가 2번 나온다.


```
# 소멸시켜봅시다.
del life
del study
```

메모리 공간에서 영원히 지워버리는 친구이다. 소멸자 두 번이 뽕뽕 나온다.

그런데

```
# 생성자에서 이름을 추가적으로 받아서 출력해봅시다.
class Person:
    def __init__(self, name):
        print("사람이 생성되었습니다.")
        self.name = name
    def __del__(self):
```

```
        print("사람이 죽었습니다.")

# 홍길동이라는 이름을 가진 hong 을 만들어봅시다.
hong = Person("홍길동")
del hong
```


결과물

사람이 생성되었습니다.

사람이 죽었습니다.

사람이 죽었습니다.

왜 두 번 죽었지...?

```
cr = Person("호날두")
cr = Person("우리흥")
```


결과물

사람이 생성되었습니다.

사람이 생성되었습니다.

사람이 죽었습니다.

갑자기 왜 뒤졌지...?

cr이 호날두를 잡고 있다가, cr이 우리흥을 잡는 순간 날두는? 영원이 우리 컴퓨터 메모리 컴퓨터속에 유영하는 친구가 된 것이다. 메모리 공간 속에 누군가 유영하게 되면 파이썬은 똑똑하게도 아무도 챙기지 않는 미아인 친구구나 너는 뒤져야겠다. 총 쏘고 시체를 쓸어간다. 그래서 이걸 쓰레기를 수집한다고 해서 garbage collector라고 부른다. 자바에서도 똑같은 이름으로 부른다. 메모리를 효율적으로 쓰기 위해 포인팅 되지 않은, 할당되지 않은 값, 원래 있던 포인팅이 풀려버린 값은 치워버린다. 그래서 걔가 사라지면서 소멸자까지 작동하게 되는 것이다.

나: 그러면 소멸시키면 그냥 뿅 사라지는게 아니라 사라지는 순간 __del__이 있는지 추가적으로 확인하는 작업이 들어가있겠네? 다른 언어도 포인팅 풀렸는지 어떻게 알지????

클래스 변수/ 인스턴스 변수

홈
클래스 변수는 모든 인스턴스가 공유하는 것이다.
인스턴스 변수는 인스턴스 별로 각각 가지는 변수이다.

방명록

```
# 위의 생성자와 인사하는 메소드를 만들어봅시다.
class Person:
    name = "john"           # 클래스 변수 : 모든 인스턴스가 공유함.
    population = 0
    def __init__(self, name = "no name"):
        # 인스턴스 변수 : 인스턴스별로 각각 가지는 변수
        self.name = name
        Person.population += 1
        print(f'인구가 증가하여 {Person.population}명이 되었습니다.')
        # print('생성될 때 자동으로 호출되는 메서드입니다.')

    def __del__(self):
        Person.population -= 1
        print(f'인구가 감소하여 {Person.population}명이 되었습니다.')
        # print('소멸될 때 자동으로 호출되는 메서드입니다.')

Person()
```

Total 4,037

웹 서비스에 데이터로 저장되는 사람이 user가 있다. user를 한 번 상상해보면, 그 user들이 있고, 그 user들의 attribute가 쭉 생길 것이다. user가 몇 명 있는지를 이렇게 클래스 변수로 사용할 수 있을 것이다. 그래서 우리는 앞으로 객체를 이용해서 쓸 거고, 장고를 보면 이런게 넘쳐난다. 장고 클래스도 활용하고 직접 클래스도 만들게 될 것이다.

나: 변수만 인스턴스 변수가 생기고, 메소드는 클래스를 타고 들어가서 보는 줄 알았는데? 클래스의 함수 내용을 변경했더니 소멸될 때 이전 함수에 있었던 print 가 나왔다. 혹시 클래스도 변경한 순간 새 클래스가 생긴 것일까?

정적 메서드 / 클래스 메서드

메소드도 비슷한 요소가 있다. 인스턴스에서만 활용할 수 있는, 인스턴스들이 각각 가지고 있는 메소드가 있고, 클래스에 속한 메소드가 있다. 클래스 메서드와 인스턴스 메서드가 있다. 정적 메서드라는 제 3자가 나왔는데, 잠깐 잊고 나중에 아 이런게 있구나 하면 된다.

나중에 OOP의 대원칙을 보게 될 것이다. 데이터를 밖으로 보여주지 않는 것이다. (데이터 은닉) 직접 접근하는 Dog.num_of_dogs 이런 식으로 쓰는 것을 기피하게 될 것이다. 접근도 하나의 메소드를 만들어서 접근하게 될 것이다.

```
class Dog:
    num_of_dogs = 0
    def __init__(self, name, age):
        self.name = name
        self.age = age
        Dog.num_of_dogs += 1
    def bark(self):
        print(f"멍멍, 저는 {self.name}, {self.age}살 입니다.")
    def info(): # self를 넣지 않은 메소드, 이름 공간의 하나의 메소드처럼 작용한다. -> static method
        print("강아지입니다.")
puppy = Dog("멍멍이", 1)
Dog.info() # 아무 인자 없는데 출력이 나온다!
```


static method는 이 클래스가 공유할 일반적인 함수다. 원래 따로 둬도 상관없다. 클래스에 대한 아주 전반적인 정보나 broad하게 이 객체들이 나중에 활용할 수도 있는 경우 쓰게 된다. 정확하게 @staticmethod라고 써주는게 관례이다.

```
class Dog:
    num_of_dogs = 0
    def __init__(self, name, age):
        self.name = name
        self.age = age
        Dog.num_of_dogs += 1
    def bark(self):
        print(f"멍멍, 저는 {self.name}, {self.age}살 입니다.")

    @staticmethod
    def info(): # self를 넣지 않은 메소드, 이름 공간의 하나의 메소드처럼 작용한다. -> static method
        print("강아지입니다.")
```

@는 데코레이터라고 부를 건데, @classmethod 를 써주면 클래스 메소드가 된다. 뒤에 cls가 중요하다. class는 예약어라서 못 쓰니까 self처럼 무언가를 넣는 것이다.

```
@classmethod
def methodname(cls):
    codeblock
```

여기까지 인스턴스 메서드와 클래스 메서드의 개념을 정리해본다.

인스턴스 메서드

첫번째 인자로 인스턴스를 받는 메서드

```
class Person:
    # 인스턴스 메서드
    def greeting(self):
```

클래스 메서드

첫번째 인자로 클래스를 받는 메서드

```
class Person:
    # 클래스 메서드
    def count(cls):
```


@classmethod를 붙이면, Dog.count() 이렇게 빈칸으로만 써도 첫번째 칸에 알아서 class를 알아서 넣어주는 것이다. 이전에 우리가 bark()로만 인스턴스 메소드를 부를 수 있었던 이유도 인스턴스 메서드라면 첫번째 인자로 인스턴스 객체를 넣어주겠다는 뜻이기 때문이다.

```
class Dog:
    num_of_dogs = 0
    def __init__(self, name, age):
        self.name = name
        self.age = age
        Dog.num_of_dogs += 1
    def bark(self):
        print(f"멍멍, 저는 {self.name}, {self.age}살 입니다.")
```

```
    @classmethod
    def count(klass): # 이름이 꼭 cls일 필요는 없다. => 첫번째 인자로 클래스를 넣어준다. 이게 핵심.
        print(f"{klass.num_of_dogs}마리 생존중")

# 3마리를 만들어보고,
puppy = Dog("멍멍이", 1)
poodle = Dog("보송이", 3)
nurung = Dog("누렁이", 5)

Dog.count() # 오류가 난다.
```

인기포스트

Dog.bark() 이렇게 쓰면 왼쪽에 있는게 인스턴스가 아니라 못 넘기겠어요! 하고 오류가 나는 것이다. 그러면 억지로 인스턴스를 넘겨줄게, **Dog.bark(nurung)** 이렇게 넘겨줄게 하는 코드가 되는 것이다.

나: 하지만 self 자리에 이미 앞에 있는 것을 가져오도록 한 거면 self, nurung 이런 식으로 밀려야 하는 거 아닐까? 그런데 왜 그렇지 않지? 그렇게 프로그램을 짜서? -> 이것도 아래에서 해결, @staticmethod를 쓰면 인자의 개수를 다르게 한다고 파이썬이 알려준다.

ABOUT ME

puppy.count()를 쓸 수 있다!!!!!!!!

@staticmethod를 안 붙이면 puppy.info() 는 못한다!!!!!!!! 이건 인자로 넘겨주는 게 아무것도 없다. 객체가 들어오든, 클래스가 들어오든, 얘는 static method에 대한 정보가 없다. 나중에 통계치 잴 때 연산 같은 거 일어나면 바깥에 정의해도 되지만 이 안에서 많이 쓰는 함수들이다 해서 static method는 함수의 묶음처럼 쓰인다. **이니셜라이즈가 된 객체의 관점에서 @staticmethod가 없으면 안 돌아간다!** 데이터 접근하지 않는 일반적인 함수에 많이 쓴다.

나: 하지만 self가 들어간 애랑 안 들어간 애랑 어떻게 구별해서? 다른 애들은 접근이 가능하고, 인자 없는 함수는 접근이 안 돼서? 왜 걔는 사라지는 걸까...@staticmethod를 안 쓰면 사라지는 거라고 생각해도 되겠지.... -> 와우, 인자의 개수를 다르게 한다는 것을 정의해주는 것이다. 아래에서 해결.

스태틱(정적) 메서드 ¶

인자로 아무것도 받지 않는 메서드 (데이터 조작을 하지 않는 함수/메서드)

```
class Person:
    def info():
```

에러 메세지를 보면, add가 사라졌다기보다 cal.add(add, 5, 3) 이런 식으로 들어갔다는 것이다.

에러 메세지: add() takes 2 positional arguments but 3 were given

@staticmethod를 붙이면 첫번째 인자 무시하고 2, 3번째 인자로 연산을 시작할게라는 뜻이다.

연산자 오버로딩 (중복 정의)

+ 더하기의 원래 원초적인 기능은 오른쪽에 있는 숫자와 왼쪽에 있는 숫자를 산술적으로 더하는 것이다.

그런데 더하기 기호를 약간 곡해해서 기능을 추가해서 상식적으로 보이기는 하지만 이런 형태의 연산을 해온 것이다.

"hello" + "world"

일반적인 산술기호 말고 기능을 추가하거나 기능을 바꾸는게 오버라이딩이다.

새로 정의해서 쓰려면 less than or equal to/ equal /not equal /greater than or equal to /greater than 이 산술 연산자들을 어떻게 쓸 수 있다.

```
class Person:
    def __init__(self, name, age, asset, height, gpa):
        self.name = name
        self.age = age
        self.asset = asset
        self.height = height
        self.gpa = gpa
    def __gt__(self, obj): #다른 객체가 들어올 것이다. object의 줄임말 obj로 넣었다.
        if self.age > obj.age:
            return True
        else:
            return False

minsu = Person('minsu', 28, 700000, 178, 4.2)
insung = Person('insung', 38, 70000000, 189, 1.8)

minsu > insung
```

클래스에 __gt__() 가 정의되어있으면 > 를 사용할 수 있다!

```
# 이것은 실제로 다음과 동일하다.
insung.__gt__(minsu)
insung > minsu
```

c) 연산자 오버라이딩 (12)

다: 인스턴스.함수를 쓰면 자동으로 첫 번째 인자가 self가 들어가는 것! 에엥 그런데 왜 Calculator.add(5, 3) 왜 돌아갔지? self없는데?
다: 오버로딩과 오버 라이딩은 뭐가 다르지...?

이 중에서 두 개의 객체가 동일한지 판별하는 equality를 많이 사용하게 될 것이다. 구글에 python overloading list를 보면 사용 가능한 애들이 쭉 나온다.

상속

객체지향 프로그래밍을 검색해보면, 프로그래밍의 패러다임의 하나이다. 쓰는 방법 중에 하나이다. 앨런 케이가 스몰토크를 만들면서 가게되었고, 특징 부분에 가보면 자료 추상화을 배웠다. 상속을 배울 것이고, 뒤에 있는 것들은 추후에 해도 된다. 동적 바인딩은 개념 상에서 살짝 다뤘다.

웹 개발에서 OOP가 뭐냐? 면접에서 물어보는 경우가 있다. 그러면 이거는 뭐를 위해서 나온 개념이며, 이런 게 특징입니다. 간단하게 정리해서 말할 수 있어야 한다.

OOP는 바깥 세계를 인지하는 자연스러운 형태. 나무의 공통점을 모아서 하나의 분류 체계를 만들고 인지한다. 그런데 여러 분류 체계를 각각의 관계를 동떨어진 관계로 나누는게 아니라, 일정한 hierarchy를 가지게 만든다.

실제로 웹에 들어가면 다음 같은 형태의 코드를 많이 보게 될 것이다.

```
class User:
    name = "빈"
    def __init__(self, email, password, name):
        self.email = email
        self.password = password
```

```
        self.name = name
    def create_post(self, title, content):
        print(f"{self.name}님이 {title}라는 제목의 {content}를 작성하였습니다.")
user1 = User('asdf@asdf.com', '12341234', 'john')
user1.create_post('하하하 제목이지', '내 첫번째 글')

class AdminUser:
    def __init__(self, email, password, name):
        self.email = email
        self.password = password
        self.name = name
    def create_post(self, title, content):
        print(f"{self.name}님이 {title}라는 제목의 {content}를 작성하였습니다.")
    def delete_post(self):
        print("글을 삭제했습니다.")

class SuperAdminUser:
    def __init__(self, email, password, name):
        self.email = email
        self.password = password
        self.name = name
    def create_post(self, title, content):
        print(f"{self.name}님이 {title}라는 제목의 {content}를 작성하였습니다.")
    def delete_post(self):
        print("글을 삭제했습니다.")
    def delete_user(self):
        print("유저를 삭제했습니다.")
```

중복되는 부분이 더럽게 많다! 상속을 하면 특정 클래스에 들어간 내용을 고스란히 활용할 수 있게 한다.

```
class Person:
    def __init__(self, name):
        self.name = name
    def sleep(self):
```

```
        print("쿨쿨")

class Student:
    def __init__(self, name):
        self.name = name
    def sleep(self):
        print("쿨쿨")
    def study(self):
        print("열공열공")
```

안드로이드 스튜디오 (3)

Person 에 있는 것을 Student에 옮기겠다 하면, class Student(상속받고자 하는 클래스의 이름): 이렇게 적

인공지능

어주면 된다. Person 객체는 잠을 잘 수 있다. Student에는 생성자도 만들지 않았는데 잘 만들어졌다.

머신러 머신러 알 위

```
# 사람 클래스를 상속받아 학생 클래스를 만들어봅시다.
class Person:
    def __init__(self, name):
        self.name = name
    def sleep(self):
        print("쿨쿨")

class Student(Person):
    def study(self):
        print("열공열공")

john = Person('john')
john.sleep()
# john.study() # 에러! Person 객체인 john은 study라는 것을 가지고 있지 않다.

donghoon = Student('donghoon')
donghoon.sleep()
donghoon.study()
```

자식은 부모의 것을 다 할 수 있지만, 부모는 자식의 것을 할 수 없다.

```
# 진짜 상속관계인지 확인해봅시다.
issubclass(Person, Student) # False
issubclass(Student, Person) # True
```

머신러닝 (11)
오버라이딩

```
class Student(Person):
    def __init__(self, name, student_id): # 이게 오버라이딩. 자식이 이거는 직접 만들어 쓰겠다.
        self.name = name
        self.student_id = student_id
```

안가르쳐줌딩이랑 오버라이딩이랑 뭐가 달라요?

머신러 머신러 알 윈
super()

```
class Person:
    def __init__(self, name, age, number, email):
        self.name = name
        self.age = age
        self.number = number
        self.email = email

    def greeting(self):
        print(f'안녕, {self.name}')

class Student(Person):
    def __init__(self, name, age, number, email, student_id):
        self.name = name
        self.age = age
        self.number = number
        self.email = email
        self.student_id = student_id

p1 = Person('홍길동', 200, '0101231234', 'hong@gildong')
s1 = Student('학생', 20, '12312312', 'student@naver.com', '190000')
```

이렇게다 안 쓰고 부모 클래스의 내용을 사용하고 싶을 때 super()를 쓸 수 있다. 이 super()의 역할은 Person. 으로 지칭했을 때와 똑같은 역할을 해줄 것이다. 그러나 Person이라는 네임 스페이스로 접근하려면 self를 써줘야한다. super()는 그 내용까지 한꺼번에 넣어주는 것이다.

머신러닝 (11)

```
class Student(Person):
    def __init__(self, name, age, number, email, student_id):
        super().__init__(name, age, number, email)
        self.student_id = student_id

class Student(Person):
    def __init__(self, name, age, number, email, student_id):
        Person.__init__(self, name, age, number, email)
        self.student_id = student_id

p1 = Person('홍길동', 200, '0101231234', 'hong@gildong')
s1 = Student('학생', 20, '12312312', 'student@naver.com', '190000')
```

ABOUT ME

부모 클래스의 메소드와 똑같은 이름의 메소드를 자식 클래스에 만들어본다.


```
# 학생은 공손하게 이야기를 해봅시다.
class Person:
    def __init__(self, name, age, number, email):
        self.name = name
        self.age = age
        self.number = number
        self.email = email

    def greeting(self):
        print(f'안녕, {self.name}')

class Student(Person):
    def __init__(self, name, age, number, email, student_id):
        super().__init__(name, age, number, email)
```

```
        self.student_id = student_id


    def greeting(self):
        print(f'안녕하세요, 저는 {self.name}입니다.')

p1 = Person('홍길동', 200, '0101231234', 'hong@gildong')
s1 = Student('학생', 20, '12312312', 'student@naver.com', '190000')

p1.greeting()
s1.greeting()

# student에 def greeting을 안 써도 돌아간다.
# 자식 클래스에 없을 경우에는 부모 클래스에 올라가서 이게 있는지 확인한다.
# 아까 탐색하는 순서랑 비슷하다!
```

상속 관계에서의 이름 공간

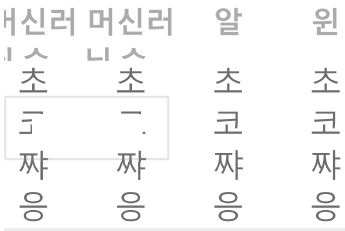
부모 클래스에 있는 greeting() 같은 것, 실제 코드가 그대로 복사된 게 아니라, 없을 때 찾아 올라가는 형태로 보면 된다. 상속하고 있는 부모 클래스까지 찾아 올라간다.

oop2.ipynb

질문 목록

- 과목 평가 범위는? 주피터로 배운 것만! 25문제
- 월말 평가 범위는? 플라스크 서버 등 배운 거 전체! 손코딩 포함
- 오브젝트는 무엇을 의미하는가? 거의 인스턴스와 동일하다. 맥락에서 다르게 부른다. 예시를 뜻할 때는 인스턴스라고 하고, 클래스에서 생성된 사물은 객체라고 한다. 그래서 맥락 상으로 객체, 오브젝트라는 말을 더 많이 쓴다.
- c언어와 달리 객체 지향이 나온 이유는 그거를 한데 묶을 수 있어서이고, 주어와 동사의 필요성이라는 구체적 설명은 단지 비유를 하신 것인가? 그렇다. 반댓말은 절차 지향적 언어인데, c언어는 자료형을 만들려다 나온 언어이기 때문에 그렇고, 사실 c의 구조체를 통해서 struct를 통해서 객체 지향을 만들 수도 있다. 객체 지향이 없었다면 데이터가 다 따로 존재했을 것이다. 나의 생일, 나이, 성별이 따로 있는데, 그것을 함수 안에 데이터를 넣어서 조작하기 쉽도록 하기 위해 한데 묶어서 편리하게 클래스로 묶어주는

홈
방명록
머신러닝 (1)
자료구조&알고리즘 (12)
TIL (17)
안드로이드 스튜디오 (3)
인기포스트



ABOUT ME
Today 89 Yesterday 92
Total 4,037

것이다. 그런데 그 때 모양새에서 두드러진 것이 객체가 무엇을 하다, 이렇게 주어 동사 형태로 되는 것이다. 어제 사용해 본 터틀도 터틀을 들고와서 꼬북이에게 많은 걸 시킬 수 있다. 우리가 직접 터틀을 만든다고 생각하면 정말 까다로울 것이다. OOP로 포장을 하는 것을 추상화 (abstraction)이다. 추상화라는 단어는 안 쓸 거긴 한데, 세상을 어떻게 요약하는가, 불필요한 것을 다 뺀다는 말로 쓸 것이다.

- Experts (1/3)은 2문제 정도 나오고 점수로 intermediate, ad1, ad2, pro가 나눠질 것이다. pro는 c++, java로만 짤 수 있다. .sort() 같은 method를 쓸 수 없고 내가 리스트를 받아서 다 sorting을 해야한다. 그 리프 expert 시험은 시간이 엄청 길다. 복잡한 문제 상황을 어떻게 코드로 잘 옮겨오는지가 문제이다. 알고리즘을 잘 해야 하는 경시대회보다는 "구현"에 초점을 맞춘다.

'TIL' 카테고리의 다른 글

SSAFY 파이썬 기초 - 10 (0)	09:36:48
SSAFY 파이썬 기초 - 9 (0)	2019.01.14
SSAFY 파이썬 기초 - 8 (0)	2019.01.11
SSAFY 파이썬 기초 - 7 (0)	2019.01.10
SSAFY 파이썬 기초 - 6 (0)	2019.01.09
SSAFY 파이썬 기초 - 5 (0)	2019.01.08

관련글 관련글 더보기

SSAFY 파이썬 기초 - 9 2019.01.14
SSAFY 파이썬 기초 - 8 2019.01.11
SSAFY 파이썬 기초 - 7 2019.01.10
SSAFY 파이썬 기초 - 6 2019.01.09

댓글 0

댓글을 입력해주세요.

등록

이전 1 2 3 4 5 … 49 다음 ›

인기포스트

인기포스트

머신러닝 수업 6주차 - Multiv..
초코쨔응

머신러닝 수업 5주차 - Parame..
초코쨔응

알고리즘 기초 유형
초코쨔응

윈도우 단축키 정리
초코쨔응

ABOUT ME

ABOUT ME

Today 89 Yesterday 92

Total 4,037

LINK

ADMIN