13기 정규세션

ToBig's 12기
조민호

Class

OOP

목표

[목표]

- OOP에 대해 이해 해봅시다
- class와 instance의 차이에 대해 이해해봅시다.
- class의 두 요소 variable과 method 그리고 magic method를 공부하고 encapsulation에 대해 느껴봅시다
- class의 상속을 보고 reusability를 느껴봅시다.

Contents

Unit 01 | Class는 갑자기..?

Q.Class를 언제 쓰는데 배우는거야??
A.이미 쓰고 있는걸요.

Unit 01 | Class는 갑자기..?

Q.Class를 언제 쓰는데 배우는거야??
A.이미 쓰고 있는걸요.

```
(base) dizweui-MacBookPro:~ dizwe$ python3
Python 3.7.3 (default, Mar 27 2019, 16:54:48)
[Clang 4.0.1 (tags/RELEASE_401/final)] :: Anaconda, Inc. on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> type(0)
<class 'int'>
>>> type("hi")
<class 'str'>
```

여러분이 python을 쓰고 있다면 모든게 클래스라고 생각해도 무방합니다.

Unit 01 | Class는 갑자기..?

Q.Class를 언제 쓰는데 배우는거야??
A.이미 쓰고 있는걸요.

sklearn.svm.SVC

class sklearn.svm. **SVC**(*C=1.0, kernel='rbf', degree=3, gamma='scale', coef0=0.0, shrinking=True, probability=False, tol=0.001, cache_size=200, class_weight=None, verbose=False, max_iter=-1, decision_function_shape='ovr', break_ties=False, random_state=None*) [source]

```
438 class SVC(BaseSVC):
439     """C-Support Vector Classification.
440
441     The implementation is based on libsvm. The fit time scales at least
442     quadratically with the number of samples and may be impractical
443     beyond tens of thousands of samples. For large datasets
444     consider using :class:`sklearn.svm.LinearSVC` or
445     :class:`sklearn.linear_model.SGDClassifier` instead, possibly after a
446     :class:`sklearn.kernel_approximation.Nystroem` transformer.
447
```

3주차에 배웠던 SVM sklearn 모델도 class로 만들어져 있습니다.
코드를 살펴봐야 할 일이 생길 때 구조를 알아야 이해하기 쉽겠죠?

Unit 01 | Class는 갑자기..?

사실 Class가 목적하는 바가 이런겁니다.
모르고도 쓰기 편하게! 멋있게 말하면 Encapsulation!

Unit 02 | OOP 개념

Class는 껍데기에 불과합니다. 먼저 Class가 탄생하게된 OOP에 대해서 알아봅시다.

OOP = Object Oriented Programming

Unit 02 | OOP 개념

Class는 껍데기에 불과합니다. 먼저 Class가 탄생하게된 OOP에 대해서 알아봅시다.

OOP = Object Oriented Programming

객체 지향 프로그래밍은 컴퓨터 프로그램을 명령어의 목록으로 보는
시각에서 벗어나
여러 개의 독립된 단위, 즉 "객체"들의 모임으로 파악하고자 하는 것이다

Unit 02 | OOP 개념

OOP = Object Oriented Programming

대상처럼 다루기를 지향하는 프로그래밍

세세하게 돌아가는 매커니즘을 몰라도 우리는 개랑 놀 수있고 게임을 할 수 있다.

Unit 02 | OOP 개념

OOP = Object Oriented Programming

물건처럼 다루기를 지향하는 프로그래밍

OOP는 패러다임 입니다. 함수를 써도 class로 만든 프로그램과 똑같이 만들 수 있는 경우가 많습니다!
예를 들어 javascript는 class로 짠걸 바로 함수로 바꿀수도 있습니다
하지만 비슷하게 구현하기 위하여 귀찮아 지는 상황이 생기기 때문에 OOP를 써보려고 하는거죠.

```
1  class tobigs {
2      manage_num: number;
3      member_num: number;
4      constructor() {
5          this.manage_num = 15;
6          this.member_num = 22;
7      }
8
9      conference_num() {
10         return this.manage_num + this.member_num;
11     }
12
13 }
```

Class 기반

```
1  "use strict";
2  var tobigs = /** @class */ (function () {
3      function tobigs() {
4          this.manage_num = 15;
5          this.member_num = 22;
6      }
7      tobigs.prototype.conference_num = function () {
8          return this.manage_num + this.member_num;
9      };
10     return tobigs;
11 }());
12
```

함수 기반

Unit 02 | OOP 개념

OOP = Object Oriented Programming

물건처럼 다루기를 지향하는 프로그래밍

Class는 OOP가 지향하는 부분을 잘 따라 올 수 있게 틀을 만들어서 오류가 안생기도록 **강제**하는 역할을 합니다.

```
package jump2java;

public class HelloWorld {
    public static void main(String args[]) {
        System.out.println("Hello World");
    }
}
```

java에서는 print hello wolrd를 할 때에도 클래스를 사용해야 합니다

Unit 03 | OOP Encapsulation과 Python Class

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 일단 python class 모양을 뜯어보며 얘기해봅시다

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
```

클래스 == 설계도

왼쪽 노란박스 같이 생긴 구조를
Class 라고 합니다.

Class는 설계도라고 생각하면 됩니다.
대상(object)를 만드려면 설계도가
있어야겠죠?

Unit 03 | OOP Encapsulation과 Python Class

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 일단 python class 모양을 뜯어보며 얘기해봅시다

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
```

설계도의 두 요소
= variable + method

variable은 객체에서 가질 데이터를 의미합니다.

python에서는 class안에 바로 적어주면 됩니다. 보통은 default로 적용할 값을 적어줍니다.

Unit 03 | OOP Encapsulation과 Python Class

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 일단 python class 모양을 뜯어보며 얘기해봅시다

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
```

설계도의 두 요소
= variable + method

method은 variable을 이용한 action을 정의합니다. class 안에 있는 variable을 이용한 함수라고 생각하시면 됩니다.

python에서는 class안에 함수를 정의하면 됩니다. 대신에 parameter에 self를 적으면 됩니다.

Unit 03 | OOP Encapsulation과 Python Class

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 일단 python class 모양을 뜯어보며 얘기해봅시다

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
```

method중에 __init__은 initialize 라는 매직 메소드(뒤에서 다룰게요)입니다.

class를 이용하여 instance로 만들 때(뒤에서 다룰게요) 자동으로 __init__함수가 실행됩니다.

쉽게 설명하면, 클래스로 처음 뭔가를 만들때 무조건 자동으로 실행되는 함수입니다.

보통은 variable을 원하는 값으로 setting할때 사용합니다.

Unit 03 | OOP Encapsulation과 Python Class

class를 이용하여 instance로 만들 때라고 했는데 무슨 소리지?
class와 instance의 차이에 대하여 알아봅시다.

Class

instance1 instance2

추상화된 개념을 구체적인 예로 설명할 때 for instance라고 하는것과 연관시키면 이해가 빠를까요...?

Unit 03 | OOP Encapsulation과 Python Class

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
allbbang = Person("얼빵이",34)
allbbang.run()
```

Person이라는 클래스를
(“투빅이”, 13) 이라는 인자를 넘긴다.

__init__이 실행되면
variable과 method가 정보가 tobigi에 담긴다.(엄밀히 말하면 메모리에 올라간다)

Unit 03 | OOP Encapsulation과 Python Class

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")

tobigi = Person("투빅이",13)
tobigi.run()
allbbang = Person("얼빵이",34)
allbbang.run()
```

tobigi와 allbbang은 같은 class에서 나왔지만 서로 다른 데이터가 들어가 구현됩니다.

이렇게 서로 다르게 구현된 tobigi와 allbbang를 각각 instance라고 부릅니다

instance끼리는 서로 다른 variable과 method를 가지게 되는거죠.

이런 개념을 encapsulation이라고 합니다.

Unit 03 | OOP Encapsulation과 Python Class

encapsulation을 통하여 데이터(variable)와 행동(method)를 합칩니다.
그리고 이것이 OOP가 “지향”하는 것이죠.
함수형으로 의미 단위마다 서로 정보를 구분하기 어려웠던 부분들을 해결해 줍니다.

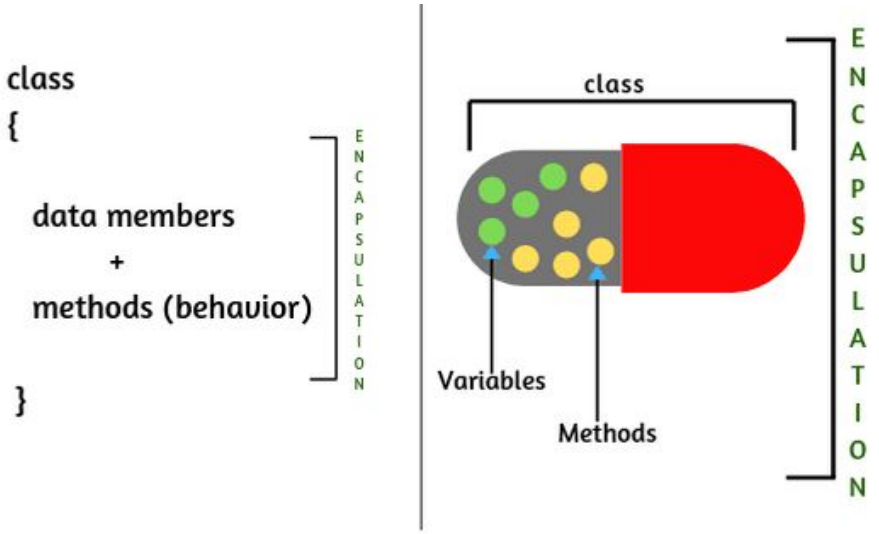


Fig: Encapsulation

```
2   name =  ""
3   age = 0
4
5   def make_person(_name, _age):
6       global name
7       global age
8       name = _name
9       age = _age
10
11  make_person("tobigi", 13)
12  print(name)
13  make_person("allbbang", 34)
14  print(name)
```

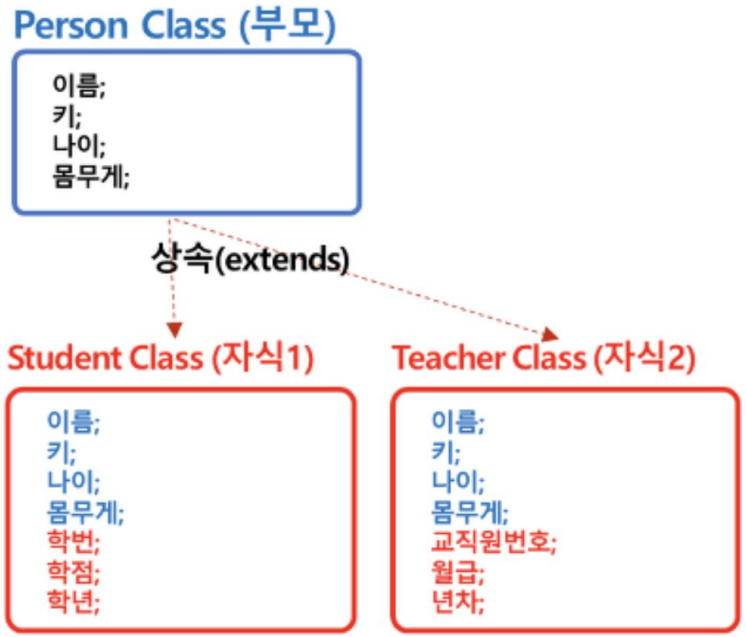
class와 비슷하게 만들면, name을 tobigi일때와 allbbang일때 구분할수가 없다.

Unit 03 | OOP Reusability와 Python Inheritance

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 두번째 Reusability

상속이란?
부모에게서 유산을 상속 받듯이 부모에게서 method(함수), variable(변수)을 그대로 가져와서 자식은 똑같은 코드를 작성할 필요 없이 사용할 수 있게 해줍니다.

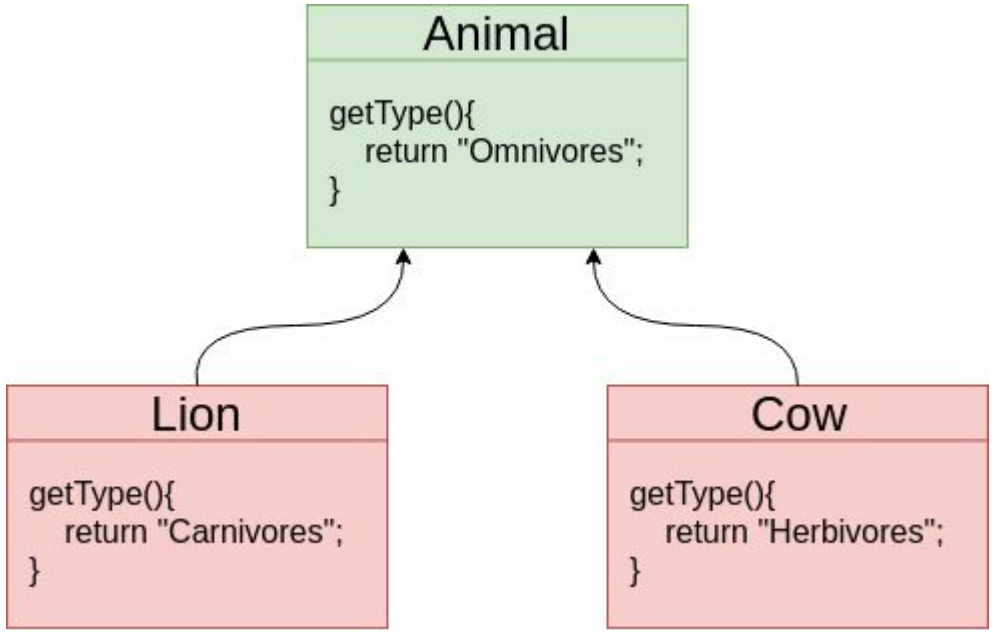
보통은 부모가 비교적 추상적인 class이고 자식이 비교적 구체적인 class입니다.



Unit 03 | OOP Reusability와 Python Inheritance

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 두번째 Reusability

Overriding이란?
부모에게서 받은 유산이 마음에 안들 때 내가 원하는대로 그대로 커스텀 하는 방식입니다.
덮어쓰기라고 생각하면 됩니다.



Unit 03 | OOP Reusability와 Python Inheritance

Reusability를 python에서 구현해 봅시다.(Person->TobigsMember)

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")
```

```python
class ToBigsMember(Person):
    siginin_year = 0
    def __init__(self, name, age, siginin_year):
        super().__init__(name,age)
        self.siginin_year = siginin_year
    def when_enter(self):
        print(str(self.siginin_year) + "에 가입")
    def study(self):
        print(self.name+"가 코딩합니다")
```

run, breath 메소드는 그대로 부모에게서 상속받고
when_enter 메소드는 자식이 자체적으로 추가하고
study 메소드는 자식이 overriding 했습니다!

Unit 03 | OOP Reusability와 Python Inheritance

Reusability를 python에서 구현해 봅시다.(Person->TobigsMember)

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")
```

```python
class ToBigsMember(Person):
    siginin_year = 0
    def __init__(self, name, age, siginin_year):
        super().__init__(name,age)
        self.siginin_year = siginin_year
    def when_enter(self):
        print(str(self.siginin_year) + "에 가입")
    def study(self):
        print(self.name+"가 코딩합니다")
```

super()는 부모의 class를 의미합니다.
super().__init__()을 통해 부모의 __init__()함수를 실행합니다.
(다른 run, breath함수들도 같은 방법으로 실행가능합니다)
부모의 init이 실행되면 name과 age가 입력한 값으로 설정됩니다.

Unit 03 | OOP Reusability와 Python Inheritance

함수보다 OOP를 쓸 때 어떤점이 좋아질까? 두번째 Reusability

```python
class Person:
    name = ""
    age = ""
    def __init__(self, name, age):
        self.name = name
        self.age = age
    def run(self):
        print(self.name+" 뛰어갑니다.")
    def breath(self):
        print(self.name+" 숨쉽니다.")
    def study(self):
        print(self.name+"가 공부합니다.")
```

```python
class ToBigsMember(Person):
    siginin_year = 0
    def __init__(self, name, age, siginin_year):
        super().__init__(name,age)
        self.siginin_year = siginin_year
    def when_enter(self):
        print(str(self.siginin_year) + "에 가입")
    def study(self):
        print(self.name+"가 코딩합니다")
```

결과적으로
자식인 TobigsMember의
instance minho는
run을 구현하지도
않았는데
부모의 run을 상속받아
실행이 가능해 졌습니다!

```python
sangwon = Person("상원",27)
sangwon.run()
minho = ToBigsMember("민호",27,2019)
minho.run()
minho.when_enter()
```

```
(tobigs) dizweui-MacBookPro:강의 자료 준비 dizwe$ python inheritance.py
상원 뛰어갑니다.
민호 뛰어갑니다.
2019에 가입
```

Unit 03 | OOP Reusability와 Python Inheritance

sklearn에서도 상속을 많이 씁니다.

```
438 class SVC(BaseSVC):
439     """C-Support Vector Classification.
440
441     The implementation is based on libsvm. The fit time scales at least
442     quadratically with the number of samples and may be impractical
443     beyond tens of thousands of samples. For large datasets
444     consider using :class:`sklearn.svm.LinearSVC` or
445     :class:`sklearn.linear_model.SGDClassifier` instead, possibly after a
446     :class:`sklearn.kernel_approximation.Nystroem` transformer.
447
```

Unit 04 | 예제로 간단히 만들기

Magic Method 란?
파이썬에서 기본적으로 고정된 기호(literal)를 클래스에서도 사용하고 싶을 때 씁니다.
파이썬의 기본적인 자료형처럼 더하기, 비교 등이 가능하게 만듭니다.

```
['__class__', '__delattr__', '__dict__', '__dir__', '__doc__', '__eq__',
'__format__', '__ge__', '__getattribute__', '__gt__', '__hash__', '__init__',
'__init_subclass__', '__le__', '__lt__', '__module__', '__ne__', '__new__',
'__reduce__', '__reduce_ex__', '__repr__', '__setattr__', '__sizeof__', '
__str__', '__subclasshook__',
'__weakref__', 'data', 'printInstance']
```

Unit 04 | 예제로 간단히 만들기

Magic Method 란?

- __init__

```
class test:
    data = 10

    def __init__(self, data):
        self.data = data
        print(self.data)



t = test(30)
print(t)
```

- __str__

```
class test:
    data = 10

    def __init__(self, data):
        self.data = data
        print(self.data)

    def __str__(self):
        return 'called by str'



t = test(30)
print(t)
```

```
30
called by str
```

- __lt__

```
class Food():
    def __init__(self, name, price):
        self.name = name
        self.price = price

    def __lt__(self, other):
        if self.price < other.price:
            return True
        else:
            return False


food_1 = Food('아이스크림', 3000)
food_2 = Food('햄버거', 5000)
food_3 = Food('콜라', 2000)

# food_2가 food_1보다 큰지 확인
print(food_1 < food_2)  # 3000 < 5000
print(food_2 < food_3)  # 5000 < 2000
```

```
True
False
```

Unit 04 | 예제로 간단히 만들기

__iter__, __next__

```python
import random

class MyDataLoader:
    def __init__(self, data):
        self.size = len(data)
        self.data = data

    # __iter__에서는 나를 반환해주고 __next__에서는 for loop가
    # 돌때마다 인쇄하고 싶은 값을 리턴합니다.
    def __iter__(self):
        self.index = 0
        return self

    def __next__(self):
        if self.index >= self.size:
            raise StopIteration # size보다 커지면 for loop 끝!

        n = self.data[self.index]
        self.index += 1
        return n

    def shuffle(self):
        random.shuffle(self.data)
```

```python
loader = MyDataLoader([i for i in range(300)])
loader.shuffle()
for i, x in enumerate(loader): # for loop 가 도니까 __iter__
    print("{} : {}".format(i,x))
```

매직 메소드를 이용하여 MyDataLoader class를 만들어서 데이터를 shuffle하고 하나씩 인쇄하는 코드를 만들어봅시다.

for loop in ~
~ 자리에 일반적으로 list가 들어오지만 instance 자체를 넣어 Interator를 만들수도 있습니다.

```
0 : 180
1 : 86
2 : 10
3 : 233
4 : 152
5 : 153
6 : 188
7 : 156
8 : 144
9 : 160
10 : 22
11 : 244
12 : 92
13 : 48
14 : 185
15 : 299
16 : 23
17 : 79
18 : 238
19 : 135
20 : 137
```

정리

- OOP의 개념
- class와 instance의 차이
- class의 두 요소 variable과 method 그리고 magic method(encapsulation)
- class의 상속(reusability)

Unit 04 | 과제설명

DataLoader 예제를 더 발전시켜서 MydataLoader2를 만들어 봅시다.
data를 하나 더 늘려서 data1과 data2를 인자로 받습니다.

1. data1과 data2의 각 value를 제곱 하는 기능을 만들어 봅시다.(make_square)
ex. data1 = [1,2,3] -> make_shuffle -> [1,4,9]

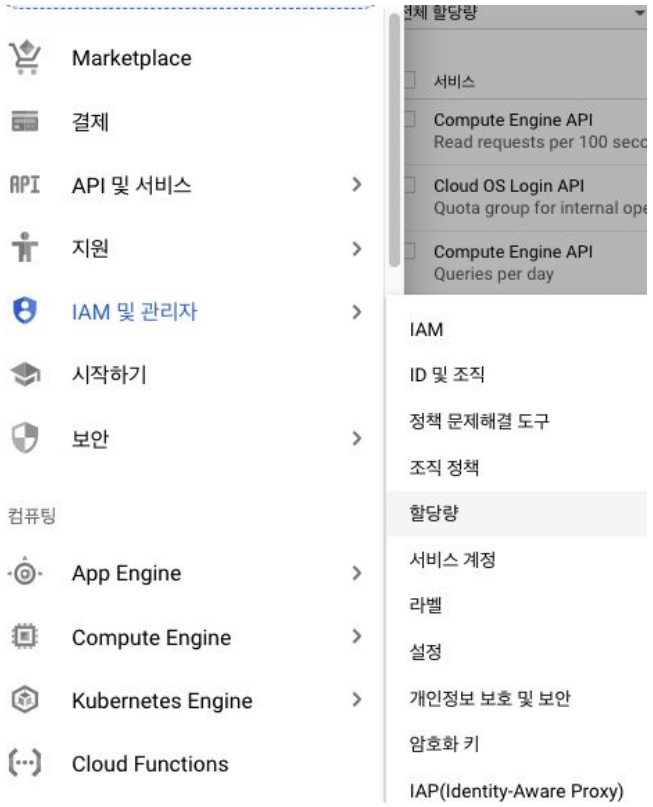
2. shuffle 기능을 data1, data2 둘다 섞을 수 있도록 바꿉니다.

3. for loop가 돌 때마다 i번째 data1의 값과 data2의 값을 더한 값을 인쇄할 수 있게 바꿔봅시다.
ex. data1 = [1,9,4] data2 = [4,1,9] -> [5,10,13]을 하나씩 리턴

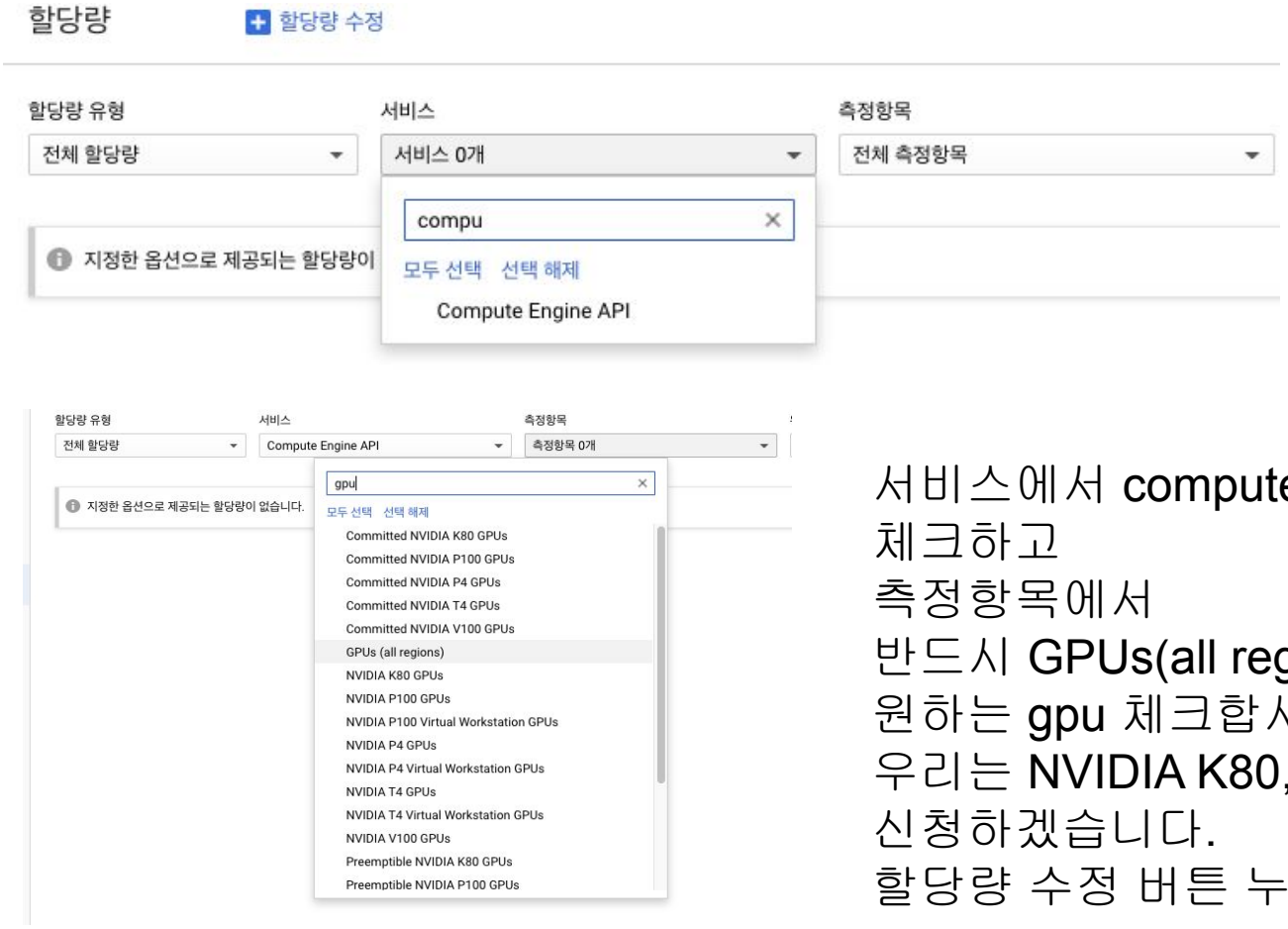
미리 GCP 신청합시다아

GCP 가입을 해봅시다. 가입은 설명 안해도 되겠죠?
https://cloud.google.com/gcp/getting-started/?hl=ko
https://zzsza.github.io/gcp/2018/01/01/gcp-intro/ 를 참고해서 가입하면 됩니다.

미리 GCP 신청합시다아

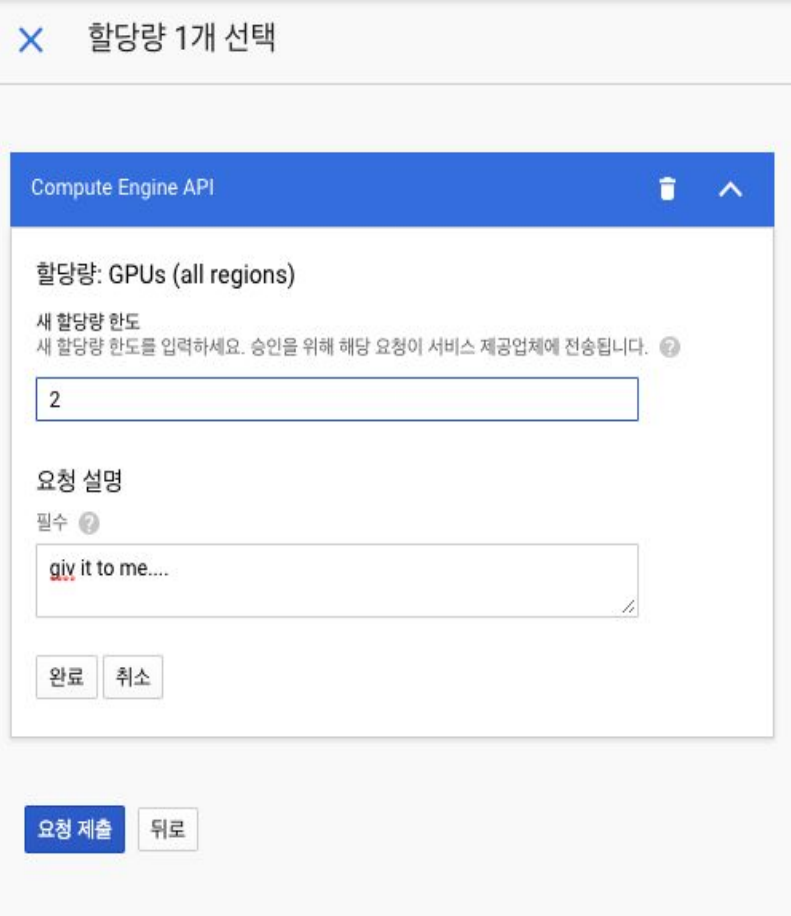


할당량 들어가귀



서비스에서 compute engine 체크하고
측정항목에서
반드시 GPUs(all regions)체크하고
원하는 gpu 체크합시다.
우리는 NVIDIA K80, P100, P4, T4 신청하겠습니다.
할당량 수정 버튼 누르기

미리 GCP 신청합시다아



quota 올려달라고 요청
(보통 2일 정도 걸립니다)

들어주셔서 감사합니다.