

Big Data Analytics Programming

Week-05. Function, Class

Jungwon Seo, 2021-Spring

배울 내용

Week-03. Python Basic II

- Python Function
- Python Class

변수(variable) = **상태(state)**



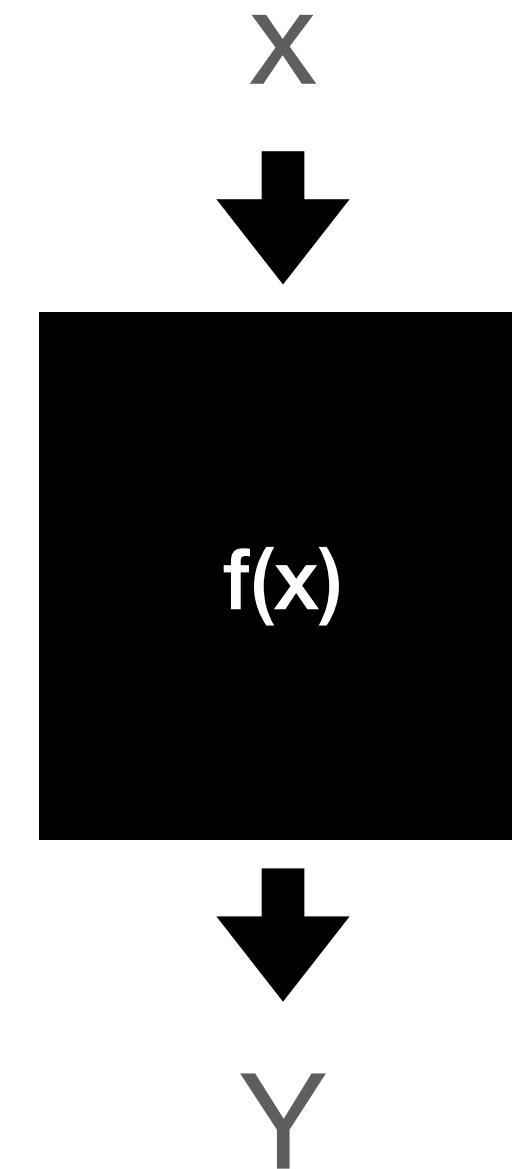
함수(function) = **행동(behavior)**



Python Function

$y=f(x)$

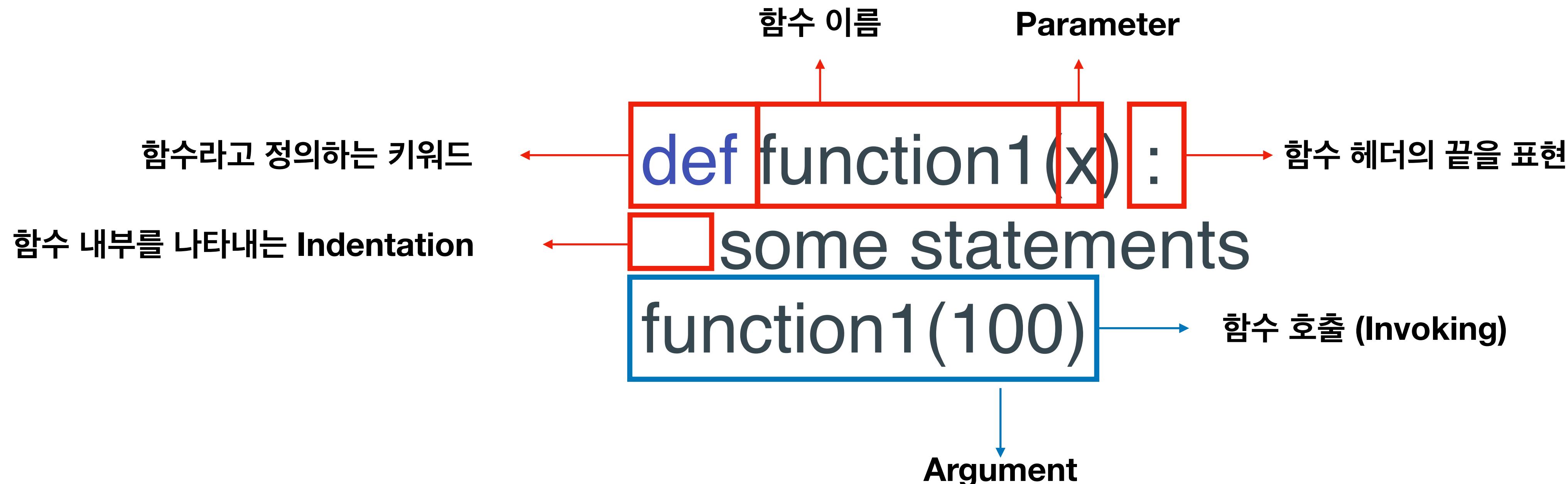
- 기본적으로 수학의 function과 비슷한 개념
 - $y=f(x)$: f라는 함수에 x라는 인자를 넣으면 y라는 값이 반환!
 - $y=2x$
- 명령/연산들을 한곳에 묶어두기 위해
 - 중복된 코드 사용 방지
 - 코드의 가독성 향상
- Python에서 함수는 keyword **def**로 시작



```
def count_char(text):  
    count = 0  
    for char in text:  
        count += 1  
    return count  
  
cnt = count_char("Hello World")  
print(cnt)
```

Python Function

함수 작성법 / 호출 법



Python Function

몇개를 입력해서 몇개를 내보낼 것인가?

- Python 함수의 구조적 고려사항
 - 매개변수(parameter)이 몇 개인가?
 - 반환값(return value)이 몇 개인가?
- 매개 변수
 - 0부터 n개 (<256) 까지 설정 가능
 - Collection 타입의 경우 1개의 변수로 인지
 - A list with 1000 elements = 1 argument
- 반환 값
 - 함수에서 연산을 끝내고 호출한 부분으로 다시 반환
 - 함수에 목적에 따라 반환 값이 없을 수도 있음

```
def func():  
    print("Nothing")
```

```
def func(a):  
    print(a)
```

```
def func(a,b,c):  
    print(a,b,c)
```

```
def func(a,b,c):  
    return 2*a, 2*b, 2*c
```

```
def func():  
    return datetime.datetime()
```

Python Function

함수 호출 (Invoking)

- **def** 로 시작해서 함수를 작성하는 것은 단지 정의 (Define) 만 하는 것!
 - Not invoking!
- 함수명과 괄호를 이용하여 호출 하여야 함
 - **def func(a,b,c)** => 정의, 실행 X
 - **func** => return id of function (address in memory)
 - **func()** => return이 있으면, return이 없으면 None
 - Q) **type(func())** 을 했을시의 타입은?

```
def func():
    return 200

func
<function __main__.func()>

print(func)
<function func at 0x7fde15b51320>

hex(id(func))
'0x7fde15b51320'

func()
200

a = func()
print(a)
200
```

Python Function

변수 영역 (Variable Scope)

- 각 변수/함수들의 유의미한 영역
- (특별한 조치를 하지 않는 한) 함수 안에서 정의 된 변수는 함수 안에서만 유효
- 8, 9, 10번의 출력 값은?



```
① def outer_func():
    ④ x = 2
    ⑤ def inner_func():
        ⑦ x = 3
        ⑧ print(x)
    ⑥ inner_func()
    ⑨ print(x)

② x = 1
③ outer_func()
⑩ print(x)
```

Python Function

Local vs Global

- 전역(Global) 변수: 코드 어디에서나 접근 가능 한 변수
- 지역(Local) 변수: 해당 영역 (Scope) 안에서만 유효한 변수
- Global Keyword: 로컬 영역에서 글로벌 영역의 변수를 수정 하기 위함

```
def test():
    print(x)
x=100
print(x) 100
test() 100
print(x) 100
```

```
def test():
    x=30
    print(x)
x=100
print(x) 100
test() 30
print(x) 100
```

```
def test():
    global x
    x=30
    print(x)
x=100
print(x) 100
test() 30
print(x) 30
```

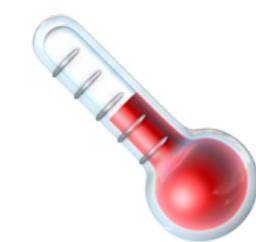
Python Function

Built-in Functions

- 개발자가 따로 정의하지 않고, 이미 Python Interpreter에서 정의 해놓은 함수들

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()

변수(variable) = **상태(state)**



함수(function) = **행동(behavior)**



클래스(class) = **객체(object)**



Python Class

Multiple Programming Paradigms

- Python은 multiple programming paradigms 언어
 - Procedural (절차지향)
 - Object-Oriented (객체지향)
 - Functional (함수형)

	절차지향	객체지향
장점	높은 가독성 쉬운 작성법	코드의 재사용성 디버깅
단점	유지보수 디버깅	설계시간
종류	C	C++, Java

Python Class

Object-Oriented Programming

- 컴퓨터 프로그램을 명령어의 목록으로 보는 것이 아닌, 여러개의 독립된 단위(객체)들의 모임으로 파악하고자 함
- OOP concepts
 - 상속 (Inheritance) : 다른 클래스(부모)의 Method 및 변수들을 상속 받을 수 있다.
 - 캡슐화 (Encapsulation) : 객체의 속성과 행위를 하나로 묶고, 그 중 일부를 외부에 감출 수 있다.
 - 추상화 (Abstraction) : 복잡한 자료, 모듈, 시스템등으로 부터 핵심적인 개념 및 기능을 간추려 나타낼 수 있다.
 - 다형성 (Polymorphism) : 한 요소에 여러 개념을 넣을 수 있다
- **Everything is an object in Python!**
 - int, float, string, dict, list, function 등 모두 사실은 Object

Python Class

Class를 배우기 전이라면..

```
player1 = {  
    "name": "손흥민",  
    "team": "토트넘",  
    "nationality": "대한민국",  
    "salary": 1000  
}  
  
player2 = {  
    "name": "호날두",  
    "team": "유벤투스",  
    "nationality": "포르투갈"  
    "salary": 2000  
}
```

```
player3 = {  
    "name": "메시",  
    "team": "바르셀로나",  
    "nationality": "아르헨티나",  
    "salary": 3000  
}
```

Python Class

Class를 배우기 전이라면..

- List에 다 정리

```
players = [player1, player2, player3]
```

- 이때, player1이 아직과 동시에 연봉이 상승

```
player[0][‘team’] = “레알마드리드”
```

```
player[0][‘salary’] = player[0][‘salary’]*2
```

- 제 2, 3의 개발자는 위의 코드를 보고 무슨 상황인지 추론을 해야함
- 만약 코드만 보고 바로 이해를 할 수 있다면?

Python Class

객체의 청사진(Blueprint)

```
class Player:  
    def __init__(self, name, team, nat, salary):  
        self.name = name  
        self.team = team  
        self.nat = nat  
        self.salary = salary  
    def update_team(self, team):  
        self.team = team  
    def update_salary(self, salary):  
        self.salary = salary
```

```
player1 = Player("손흥민", "토트넘", "대한민국", 1000)  
player1.update_team("레알 마드리드")  
player1.update_salary(2000)
```

Python Class

OOP Terms

Class

```
class Mammal:  
    def __init__(self):  
        pass
```

초기화 함수 / 생성자

(Initialization function/
Constructor)

class Human(Mammal): 부모 클래스(Parent Class)

```
def __init__(self,date_of_birth, name, nationality):
```

```
    self.date_of_birth=date_of_birth  
    self.name = name  
    self.nationality = nationality
```

Variable
(Attribute)

```
def think(self):
```

```
    pass
```

```
def move(self):
```

```
    pass
```

Function
(Method)

인스턴스화(Instantiation)

인스턴스
(Instance)

```
jungwon = Human("900302", "서중원", "대한민국")
```

Python Class

생성자 (Constructor)

- 클래스가 인스턴스 됨과 동시에 실행되는 Method
- 어떠한 객체를 만들때 초기값/세팅을 설정해주기 위함

```
class Player:  
    def set_info(self, name):  
        self.name = name  
        print("안녕! 나는 {} 이라고해!".format(name))  
  
player = Player()  
player.set_info("중원")  
print(player.name)
```

안녕! 나는 중원 이라고해!
중원

```
class Player:  
    def __init__(self, name):  
        self.name = name  
        print("안녕! 나는 {} 이라고해!".format(name))  
  
player = Player("중원")  
print(player.name)
```

안녕! 나는 중원 이라고해!
중원

Python Class

Self 변수

- 인스턴스의 Method를 호출 하면, 자동으로 자기 자신을 함수에 포함

```
class Person:  
    def __init__(self, name):  
        self.name = name  
        print(self)  
p = Person("Jungwon")  
print(p)
```

```
<__main__.Person object at 0x7f98eb84ea50>  
<__main__.Person object at 0x7f98eb84ea50>
```

```
class Person:  
    def __init__(dosentneedtobeself, name):  
        dosentneedtobeself.name = name  
        print(dosentneedtobeself)  
p = Person("Jungwon")  
print(p)
```

```
<__main__.Person object at 0x7f98eb8845d0>  
<__main__.Person object at 0x7f98eb8845d0>
```

Python Class

Variable, Method

- Class Variable / Method
- Instance Variable / Method

```
class Something:  
    class_variable = "A"  
    def class_method():  
        print("Hello I am a class method!")  
    def __init__(self):  
        self.instance_variable = "B"  
    def instance_method(self):  
        print("Hello I am an instance method!")
```

```
s = Something()  
print(s.class_variable)  
print(s.instance_variable)  
s.instance_method()  
s.class_method()  
  
A  
B  
Hello I am an instance method!  
  
-----  
TypeError Traceback (most recent call last)  
<ipython-input-49-a946649b2bec> in <module>  
      3 print(s.instance_variable)  
      4 s.instance_method()  
----> 5 s.class_method()  
  
TypeError: class_method() takes 0 positional arguments but 1 was given  
  
Something.class_method()  
print(Something.class_variable)  
try:  
    Something.instance_method()  
except Exception as e:  
    print("Error!",e)  
try:  
    print(Something.instance_variable)  
except Exception as e:  
    print("Error!",e)  
  
Hello I am a class method!  
A  
Error! instance_method() missing 1 required positional argument: 'self'  
Error! type object 'Something' has no attribute 'instance_variable'
```

Python Class

아직도 왜쓰는지 모르겠...

- URL을 Parsing하는 코드를 작성한다고 했을때.

```
class URLParser:
    def __init__(self, url):
        self.url = url
        self._parse()
    def _parse(self):
        self.protocol = url.split("://")[0]
        self.domain = url.split("://")[1].split("//")[0]
        self.query_strings = url.split("?")[1].split("&")

    def get_protocol(self):
        return self.protocol
    def get_domain(self):
        return self.domain
    def get_query_strings(self):
        return self.query_strings
    def get_query_string_with_key(self, key):
        qs_dict = {}
        for query_string in query_strings:
            k, v = query_string.split("=")
            qs_dict[k] = v

        return qs_dict.get(key, None)

url = "https://www.google.com/search?newwindow=1&sxsrf=ALeKk02Pp41fzfe2GzJT"
url_p = URLParser(url)

print(url_p.get_protocol())
print(url_p.get_domain())
print(url_p.get_query_strings())
print(url_p.get_query_string_with_key("q"))

https
www.google.com
['newwindow=1', 'sxsrf=ALeKk02Pp41fzfe2GzJTIJfu8QhaEdwM2g%3A1600176084627',
'oq=class', 'gs_lcp=CgZwc3ktYWIQAzIECAAQRzIECAAQRzIECAAQRzIECAAQRzI
QCIAQCSAQCYAQcQdnd3Mtd216yAEIwAEB', 'sclient=psy-ab', 'ved=0ahUKEwi3q0i4o
class']
```

Python Class

Use Case - Graphical User Interface

- GUI
- 테이블, 메뉴, 버튼 등을 객체화
- onTouch, onScroll 같은 사용자 액션에 대한 method 공유

virtual bool [onTouchBegan \(Touch *pTouch, Event *pEvent\)](#) override
Callback function for touch began. [More...](#)

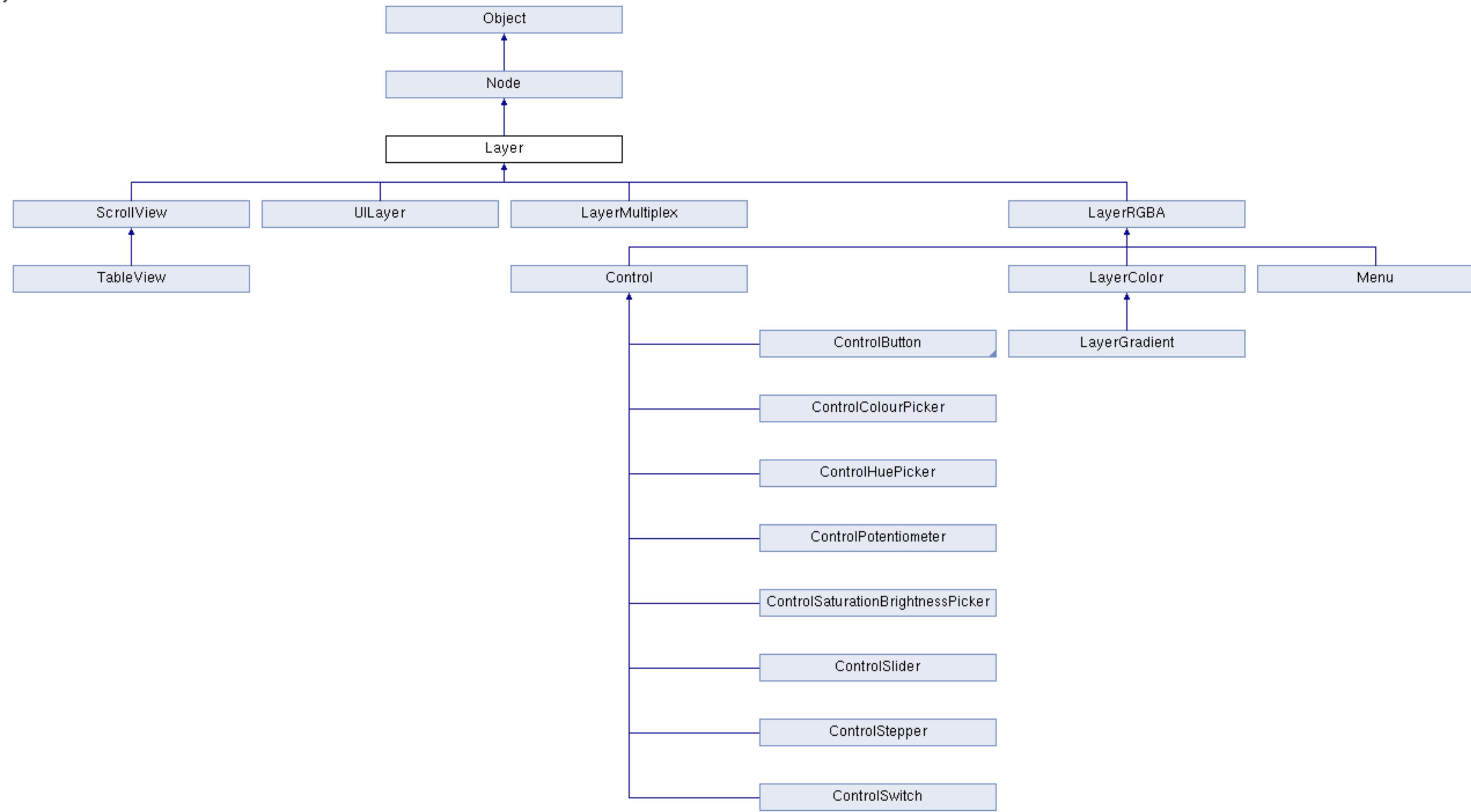
virtual void [onTouchMoved \(Touch *pTouch, Event *pEvent\)](#) override
Callback function for touch moved. [More...](#)

virtual void [onTouchEnded \(Touch *pTouch, Event *pEvent\)](#) override
Callback function for touch ended. [More...](#)

virtual void [onTouchCancelled \(Touch *pTouch, Event *pEvent\)](#) override
Callback function for touch cancelled. [More...](#)

Python Class

Use Case - Graphical User Interface



Python Class

Use Case - Game

- 게임
- 캐릭터, 건물 등을 객체화



```
class Unit:  
    def __init__(self,x,y):  
        self.cur_x = x  
        self.cur_y = y  
        pass  
  
    def move(self,new_x,new_y,speed):  
        dir_x = 1 if new_x > self.cur_x else -1  
        dir_y = 1 if new_y > self.cur_y else -1  
        while (self.cur_x,self.cur_y) != (new_x, new_y):  
            self.cur_x += dir_x*speed  
            self.cur_y += dir_y*speed  
  
    def attack(self, target):  
        pass  
  
    def hold(self):  
        pass  
  
    def patrol(self,new_x,new_y):  
        pass
```

Python Class

Use Case - Game

```
class Marine(Unit):  
    def __init__(self,x,y):  
        super().__init__(x,y)  
        self.speed = 3  
    def move(self, new_x, new_y):  
        super().move(new_x, new_y, self.speed)
```



```
class Vulture(Unit):  
    def __init__(self,x,y):  
        super().__init__(x,y)  
        self.speed = 10  
    def move(self, new_x, new_y):  
        super().move(new_x, new_y, self.speed)
```



Python Class

Use Case - Database

- ORM : Object-relational mapping
- 관계형 데이터베이스와 파이썬의 OOP성질을 연동
- 테이블 = 클래스, 컬럼 = 변수

```
class User(db.Model):  
    id = db.Column(db.Integer, primary_key=True)  
    email = db.Column(db.String(120), unique=True, index = True)  
    password = db.Column(db.String(120), nullable=False)  
    created_time = db.Column(db.DateTime, default=datetime.datetime.utcnow)  
    updated_time = db.Column(db.DateTime, default=datetime.datetime.utcnow)
```

Python Class

Use Case - Many Other Frameworks

- Framework

- Web/App Development, Deep Learning과 같이 사전에 정의/구현 되어야 하는 요소들이 많은 개발의 경우 미리 잘 짜여진 틀 안에서 개발을 진행 (파워포인트 템플릿)

```
1 import bcrypt
2
3 #from django.contrib.auth.models import User
4 from rest_framework import serializers
5
6 from .models import User, Document
7
8
9 class UserSerializer(serializers.ModelSerializer):
10     class Meta:
11         model = User
12         fields = '__all__'
13     #    URL_FIELD_NAME = 'newurl'
14
15 class DocumentSerializer(serializers.ModelSerializer):
16     class Meta:
17         model = Document
18         fields = '__all__'
```

```
1 from torch import nn
2 import torch
3
4 class BaseLine(nn.Module):
5     def __init__(self, hidden_dim, filter_size, dropout_rate, vocab_size, embedding_dim, pre_trained_embedding=None):
6         super().__init__()
7
8         self.hidden_dim = hidden_dim
9         self.filter_size = filter_size
10        self.dropout_rate = dropout_rate
11        self.embedding_dim = embedding_dim
12
13        if pre_trained_embedding is None:
14            self.vocab_size = vocab_size
15            self.embedding = nn.Embedding(self.vocab_size, self.embedding_dim, padding_idx=0)
16        else:
17            self.embedding = nn.Embedding.from_pretrained(pre_trained_embedding, freeze=False, padding_idx=0)
18            self.relu = nn.ReLU()
19            self.dropout = nn.Dropout(self.dropout_rate)
20            self.conv1d = nn.Conv1d(self.embedding_dim, self.hidden_dim, self.filter_size)
21            self.bi_rnn = nn.LSTM(self.hidden_dim, int(self.hidden_dim / 2), batch_first=True, bidirectional=True)
22            self.uni_rnn = nn.LSTM(self.hidden_dim, self.hidden_dim, batch_first=True)
23            self.max_pool = nn.AdaptiveAvgPool2d((1, self.hidden_dim))
24            self.linear = nn.Linear(self.hidden_dim, 1)
25            self.sigmoid = nn.Sigmoid()
26
27    def forward(self, x):
28        x = self.embedding(x).transpose(0, 1).transpose(1, 2)
29        x = self.conv1d(x).transpose(1, 2).transpose(0, 1)
30        x = self.relu(x)
31        x = self.dropout(x)
32        x_res = x
33        x, _ = self.bi_rnn(x)
34        x, _ = self.uni_rnn(x + x_res)
35        x = self.dropout(x)
36        x, _ = torch.max(x, 0)
37        x = self.linear(x)
38        x = self.sigmoid(x).squeeze()
39
40    return x
```

클래스를 쓴다는 것에 대한 오해

- 프로그램이 더 빠르다. (X)
- 알고리즘이 더 쉽게 구현된다. (X)
- 설계를 잘 못 했을 시, 오히려 가독성/사용성이 떨어질 수도 있다.
- 어떤 대상을 객체화하는 과정을 머리속으로 훈련!

E.O.D