

K-NET DEVELOP #7

Hyungyu Kim

K-NET

2024-06-04

시작에 앞서서...

- 우리가 직접 유연하고 재사용성이 좋은 모듈(프로그램의 부품)을 만들려면 파이썬 파일 내에서 함수, 변수들을 어떻게 관리해야 할까요?
- 힌트: 이번 시간에 다룰 **클래스**와 관계 있습니다.

```
class BankAccount:
    def __init__(self):
        self.__balance = 0

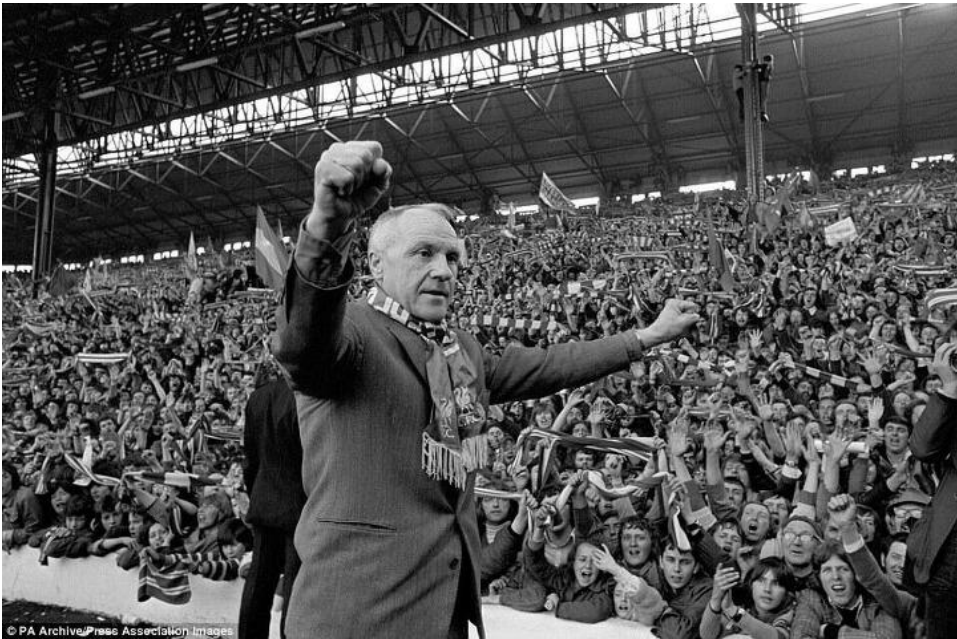
    def withdraw(self, amount):
        self.__balance -= amount
        print("통장에", amount, "원이 출금되었음")
        return self.__balance

    def deposit(self, amount):
        self.__balance += amount
        print("통장에서", amount, "원이 입금되었음")
        return self.__balance

a = BankAccount()
balance = a.deposit(100)
print("잔액은 ", balance, "입니다.")
balance = a.withdraw(10)
print("잔액은 ", balance, "입니다.")
```

시작에 앞서서...

- 파이썬의 클래스를 가지고 할 수 있는 말이 아주 많은데, 이번 시간 안에 다 하기란 힘이 들어서 저도 아쉽군요.
- 오늘의 결론, “**폼은 일시적이지만 클래스는 영원하다**”라는 사실입니다.
- 이게 무슨 말이냐고요? 그건 나중에~




**FORM IS TEMPORARY
CLASS IS PERMANENT**
★★★★★



시작에 앞서서...

- 이번 시간은 시험이 바로 코앞이죠?
- 날도 더워지는데, 우리 세미나보다도 건강(그리고 시험)이 우선입니다!
- 오늘은 지식 전달보다 감상하기 좋은 내용들을 제시하겠습니다.

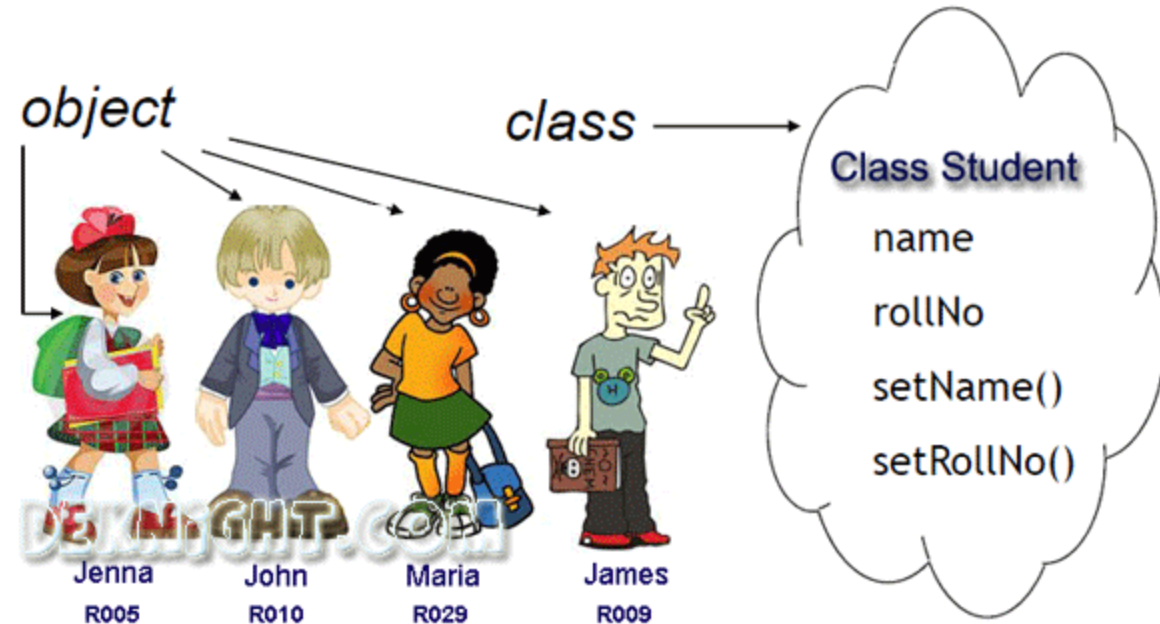


지난 시간에 이어서...

- 변수, 배열, 리스트, 딕셔너리, 조건문, 반복문, 함수, ...
- 파이썬의 모든 것은 객체입니다! (정수, 리스트, 문자열, 함수까지도...)
- 이제는 우리가 클래스를 통해 새로운 객체를 만들어보겠습니다.
- 근데... 제가 아직까지도 그 '객체'(Object)가 뭔지 설명하지 않았죠?

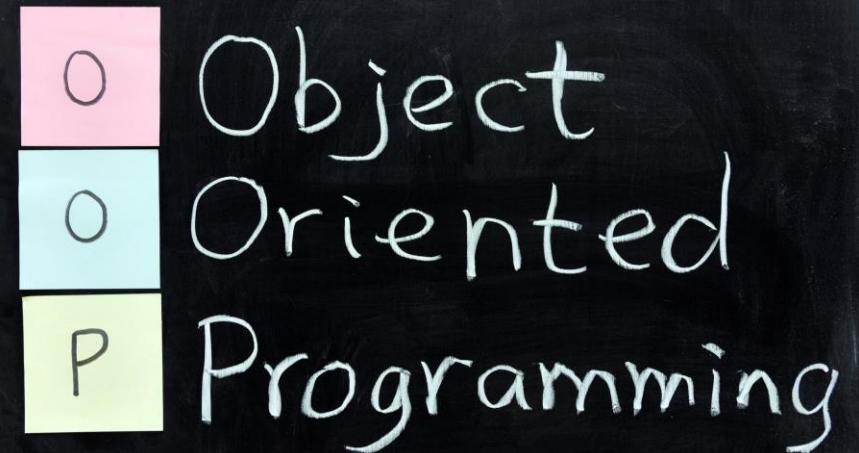
이제서야 등장하는 객체의 의미

- 객체: 데이터와 행위를 하나로 묶어 나타내는 기본 단위
(사실, 이거 저도 정의를 외우지 않아서 Gemini한테 물어봤어요...)
- 객체는 **현실 세계의 사물, 개념을 프로그램 내에서 표현**할 수 있는 효과적 수단이라고 생각할 수 있습니다.



이제서야 등장하는 객체의 의미

- 객체 지향 프로그래밍(Object Oriented Programming): 객체를 기본 단위로 하여, 객체 간의 상호작용으로 프로그래밍하는 방식
- 파이썬, 자바, C++ 등은 객체 지향 프로그래밍을 지원하는 언어입니다.
- 뭔가 설명이 따분하지만 자주 쓰이는 용어입니다.
- 저를 믿으신다면, 이 용어에 눈도장을 찍어주세요…!



객체가 갖는 특징

- 데이터(속성)가 있습니다!
드럽게 진부한 예시: 사람은 나이, 이름, 키, 국적 등이 있습니다.
- 행위(메서드)가 있습니다!
또 다시 진부한 예시: 사람은 말하기, 걷기, 먹기 등을 할 수 있습니다.
- 중요: 객체는 '클래스'라는 틀 안에서 만들어집니다.
- 그래서 클래스는 설계도 역할을 하며, C언어의 구조체와 비슷한 면이 있습니다.
- 리스트 등도 `type()`을 씌워 출력해보면 `<class 'list'>`라고 나옵니다. (진짜임)

```
1 print(type([1, 2, 3]))
```

```
<class 'list'>
```

```
1 print(type(123))
```

```
<class 'int'>
```


클래스로 객체 만들기

- 클래스를 가지고 객체를 만들 수 있습니다.
- 특히, 클래스를 통해 만든 객체는 보통 **인스턴스**(instance)라고 부르는 편입니다.
- 객체에는 **속성**, **메서드**(함수) 등이 있으니, 모듈 작성 시에도 클래스를 쓰면 유리합니다!
- 이는 지난 시간의 질문에 대한 한 가지 답이 될 수 있습니다.

클래스로 객체 만들기

- 참고: 그동안, 리스트에서 함수를 불러올 때 `객체.함수()` 형태로 **마침표**를 찍어 메서드를 불러왔죠? 예를 들어, `list.append(item)`라거나...
- 클래스로 만든 객체에서도 똑같이 할 수 있습니다! 간단한 예시를 몇 개 보여드리자면...
- 참고: `__init__` 처럼 양옆에 `__` (언더스코어 2개)가 붙은 함수는 특수한 함수들입니다. (이걸로 `len()`, 덧셈 연산자, 인덱싱 등을 지원할 수 있습니다. 자세한 것은 검색을...)

```
1 # 보통 파이썬 책에 있을 것 같은 쉬운 예시
```

```
2
```

```
3 class KPopGroup:
```

```
4     def __init__(self, name): # "생성자(constructor) 메서드"입니다.
```

```
5         # self라고 적은 것은 이 클래스가 생성하는 객체(instance)를 뜻합니다.
```

```
6         self.name = name # self.변수 형태로 사용합니다.
```

```
7
```

```
8     def greet(self):
```

```
9         print(f"안녕하세요, {self.name}입니다!")
```

```
1  # 생성자를 호출해 인스턴스 생성
```

```
2  # 위의 코드에서 __init__은 인스턴스 생성 시 기본적으로 실행되는 코드를 써주는 곳입니다.
```

```
3  bts = KPopGroup("BTS")
```

```
4
```

```
5  bts.greet() # 객체의 메서드 호출
```

```
안녕하세요, BTS입니다!
```

클래스로 객체 만들기

- 1주차 때 보여드렸던 코드인데, 기억나나요?
- 이 코드가 실행되게끔 클래스를 구현해볼까요?



```
hyungyu = Hyungyu()  
  
if hyungyu.is_from_korea():  
    print("안녕 현규야!")  
  
if "t" in hyungyu.mbti() or "T" in hyungyu.mbti():  
    print("너 T야?")
```

클래스로 객체 만들기

- 멤버 변수 이름, 클래스 이름, 함수(메서드) 이름 등을 잘 적어줘야 합니다.

```
class Hyungyu:
    def __init__(self, nationality="Korea", mbti="ENTP"):
        self.nationality = nationality
        self.mbti = mbti

    def is_from_korea(self):
        # 괄호 안에 self를 넣어줘야 hyungyu.is_from_korea() 형태로 쓸 수 있습니다.
        code_list = ["ko", "kor", "korea", "korea"]
        return self.nationality.lower() in code_list

    def get_mbti(self): # 위의 "mbti" 대신 get_mbti(mbti 가져오기)라는 이름으로 바꿨습니다.
        return self.mbti
```

클래스로 객체 만들기

- 잘 작동합니다! 나중에 다시 보실 때, 이쯤에서 1장 슬라이드를 꼭 다시 펼쳐보세요.
- 그리고 떠올려봐요! 우리가 얼마나 멀리까지 왔는지...

```
1  hyungyu = Hyungyu() # 객체(인스턴스) 생성
2
3  if hyungyu.is_from_korea(): # 객체로부터 메서드 불러오기
4      print("안녕 현규야!")
5
6  if "t" in hyungyu.get_mbti() or "T" in hyungyu.get_mbti():
7      print("너 T야?")
```

```
안녕 현규야!
너 T야?
```

클래스로 자료형 만들기?

- 참고: 클래스를 만드는 것은 자료형을 새로 만드는 것과 동일한 효과를 냅니다!
- 아까 만든 Hyungyu 클래스 안에는 생성자를 제외하면 변수 2개(nationality, mbti), 함수 2개(is_from_korea, get_mbti)가 묶여 있어요.
- 여러분, 다들 저를 믿으시죠?????

```
1 print(type(hyungyu))
```

```
<class '__main__.Hyungyu'>
```

클래스로 계산기 만들기

```
1  fc = FourCal(5, 4)
2
3  print(fc.div()) # 5 / 4 == 1.25
```

1.25

- 점프 투 파이썬에 있는 사칙연산 계산기 예시입니다.
- 예시가 좋긴 한데, “감상”하기에는 좀 시시한 면이 있죠?

```
class FourCal:
    def __init__(self, first, second):
        self.first = first
        self.second = second

    def setdata(self, first, second):
        self.first = first
        self.second = second

    def add(self):
        result = self.first + self.second
        return result

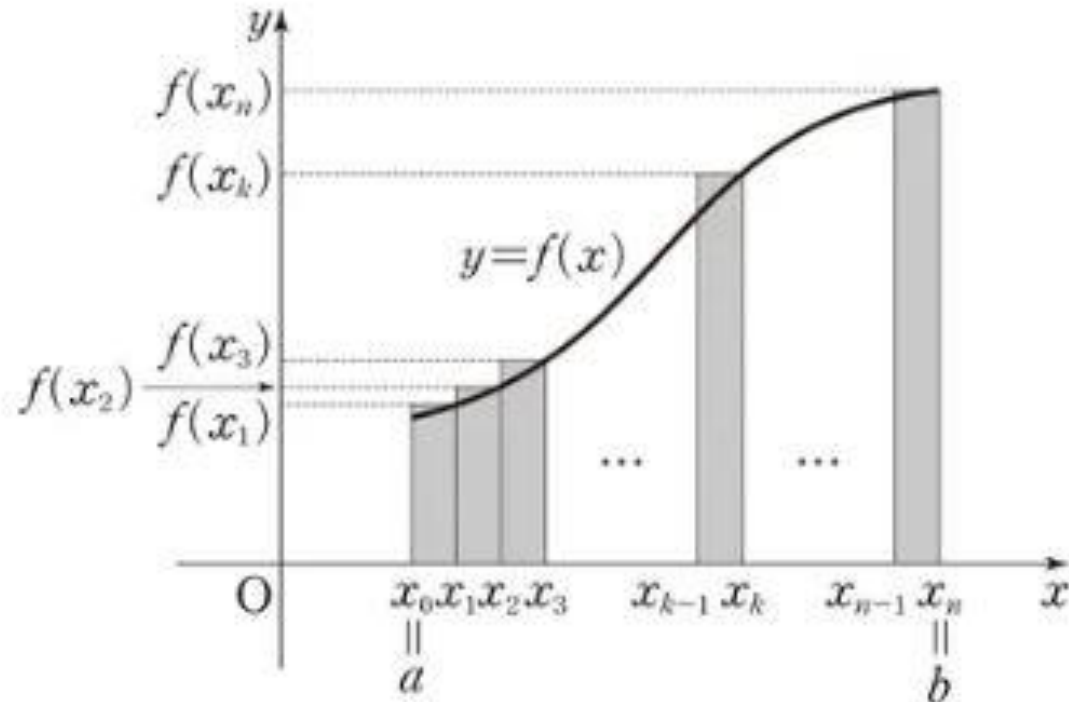
    def mul(self):
        result = self.first * self.second
        return result

    def sub(self):
        result = self.first - self.second
        return result

    def div(self):
        result = self.first / self.second
        return result
```


클래스로 계산기 만들기

- 뜯금없지만 우리는 좀 덜 시시하게 적분 계산기를 만들어봅시다.
- 네? 적분이요???



화석 김현규의 끈대질 시간...

- 아까의 코드는 정적분의 정의에서 극한 대신 소구간의 개수를 크게 만든 예시입니다.
- 대학수학1 시간에 배운 정적분의 정의를 기억하시는지? (리만 합의 극한)

$$\int_a^b f(x) dx \stackrel{\text{def}}{=} \lim_{n \rightarrow \infty} \sum_{k=1}^n f(x_k^*) \Delta x_k$$

여기서 $P = \{a = x_0, x_1, x_2, \dots, x_{n-1}, x_n = b\}$ 는 구간 $[a, b]$ 의 분할이고,

각 x_k^* 는 $[x_{k-1}, x_k]$ 이며 $\Delta x_k = x_k - x_{k-1}$ ($k \geq 1$)입니다.

여기서 분할 P 가 $[a, b]$ 의 n 등분이면, (위의 예시가 꼭 n 등분일 필요는 없습니다.)

화석 김현규의 끈대질 시간...

- 여기서 분할 P 가 $[a, b]$ 의 n 등분이면 $\Delta x_k = \frac{b-a}{n}$ 이며 각 $x_k = a + \frac{b-a}{n}k$ 가 됩니다.
- 그리고 각 x_k^* 가 $x_k^* = x_k$ 이면 아까의 리만 합의 극한 식은

$$\int_a^b f(x) dx \stackrel{\text{def}}{=} \lim_{n \rightarrow \infty} \sum_{k=1}^n f(x_k^*) \Delta x_k = \lim_{n \rightarrow \infty} \sum_{k=1}^n f\left(a + \frac{b-a}{n}k\right) \frac{b-a}{n}$$

- 오른쪽 식에서 극한 대신 n 에 상대적으로 큰 수를 대입하는 근사 방식을 사용합니다.
- 우리가 배운 내용들로 구현해보겠습니다.

클래스로 계산기 만들기

- 구현 예시
- 참고: 여기서 `__init__`은 생성자 호출 시 처음 한 번만 실행됩니다.
(`set_integrand` 메서드가 존재하는 이유)

```
class Integral:
    """
    적분 계산을 위한 클래스입니다. f는 피적분 함수입니다.
    """
    def __init__(self, f):
        self.f = f # integrand

    def set_integrand(self, f):
        self.f = f
```

클래스로 계산기 만들기

- 구현 예시

```
def integral(self, a, b, N=10000):  
    '''  
    적분 구간을 아주 잘게 쪼개서 컴퓨터의 힘으로 근사적으로 적분을 계산 (리만 합의 극한)  
    '''  
  
    h = (b-a) / N # 적분 구간을 N = 10000등분하여 계산하되  
    if h > 0.01: # 적분 구간이 너무 넓을 시 소구간 길이를 작은 상수 0.01로 강제하기  
        h = 0.01  
        N = int((b-a) / h)  
    P = [a + i*h for i in range(N)]  
    riemann_sum = 0  
    for i in range(N):  
        x_i = P[i]  
        riemann_sum += h * self.f(x_i) # 적분 근사 계산  
    return riemann_sum
```

클래스로 계산기 만들기

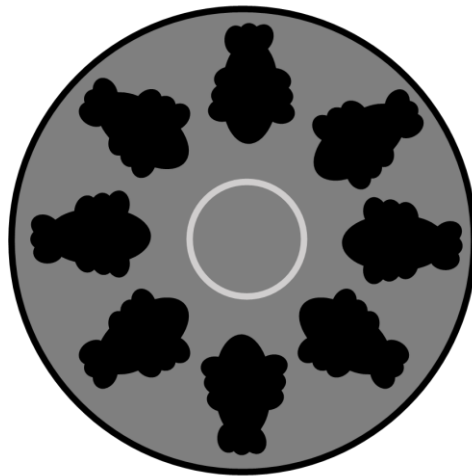
- 클래스 사용 예시: $\int_0^1 x^2 dx = 1/3 = 0,333 \dots$
- 약간의 오차가 있지만 그래도 잘 나오는 것 같죠?

```
1  # 클래스 생성자 불러오기 (인스턴스화)
2  integral = Integral(lambda x: x**2) # 피적분함수를 y=x^2로 설정
3
4  # y=x^2를 0부터 1까지 적분
5  res = integral.integral(0, 1)
6
7  # 약 1/3 == 0.333...이 나와야 함
8  print(f"0부터 1까지 적분한 결과는 {res:.6f}입니다.")
```

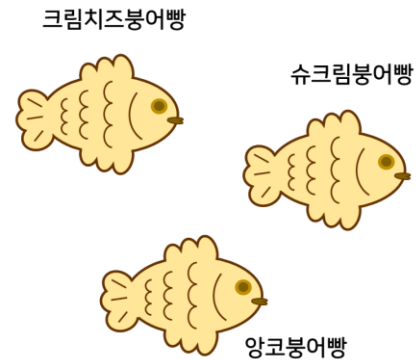
0부터 1까지 적분한 결과는 0.333283입니다.

무한히 객체를 만드는 기계

- 클래스를 통해 서로 다른 객체를 여러 개 만들어내고 삭제도 할 수 있습니다.
- 객체가 붕어빵이면, 클래스가 붕어빵 틀인 느낌?
- 프로그래밍 책에서 정말 자주 제시하는 예시라서, 여러분도 알고 계시면 좋겠습니다.
- 역시, 객체의 모습(폼)은 일시적이지만 클래스(class)는 영원한 법이군요!



붕어빵 틀은 클래스



붕어빵은 객체

무한히 객체를 만드는 기계

- 참고: 보통 log라고 쓰면 **자연로그**를 의미하는 경우가 많습니다. (때에 따라 상용로그...)
- 참고: 파이썬은 ‘**인터프리터 언어**’라서, 코드에 에러가 있어도 결과를 한 줄 한 줄 실행하다가 도중에 에러를 냅니다. (그러면, **컴파일 언어**의 경우는?)

```
1  # 클래스를 통해 객체를 여러 개 만들고 지울 수 있습니다.
2
3  import numpy as np
4
5  int_1 = Integral(lambda x: np.sin(x/2)) # 사인 함수
6  res_1 = int_1.integral(0, np.pi)
7  del int_1
8
9  int_2 = Integral(lambda x: abs(x)) # 절대값 함수
10 res_2 = int_2.integral(-1, 1)
11 del int_2
12
13 int_3 = Integral(lambda x: np.log(x)) # 로그 함수
14 res_3 = int_3.integral(1, np.e)
15 del int_3
16
17 print(f"{res_1:.4f}, {res_2:.4f}, {res_3:.4f}")
18
19 print(int_1, int_2, int_3)
```

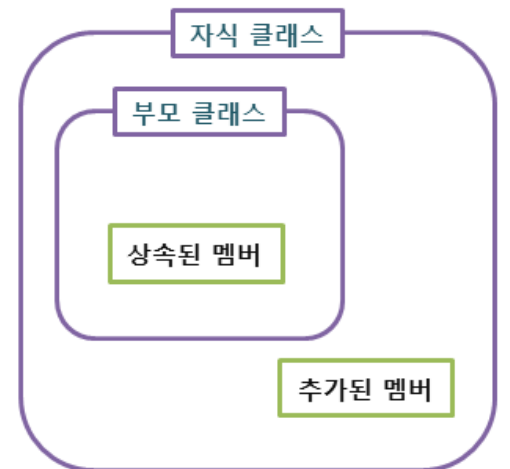
```
1.9998, 1.0000, 0.9999
```

```
-----
NameError                                Traceback (most recent call last)
<ipython-input-83-522e562fdb03> in <cell line: 19>()
    17 print(f"{res_1:.4f}, {res_2:.4f}, {res_3:.4f}")
    18
--> 19 print(int_1, int_2, int_3)

NameError: name 'int_1' is not defined
```

클래스의 상속

- 이전 클래스의 기능들을 물려받아서 더 확장된 클래스를 만드는 것을 상속이라 합니다.
- 기존 클래스를 수정하지 않고도 클래스를 확장할 수 있습니다.
(남이 만든 numpy 라이브러리 등은 직접 수정이 어렵겠죠?)
- 예전의 클래스와 새로운 클래스를 각각 부모 클래스, 자식 클래스라고 부릅니다.
- 클래스의 상속은 간단히만 다루겠습니다. 봤을 때 이게 뭔지 모르지 않을 정도로만...



클래스의 상속

- 아까 만든 KPopGroup이라는 클래스를 상속하여 BTS 클래스를 만드는 예시입니다.
- 여기서 `super()`는 부모 클래스를 의미하고, `super().__init__()`은 부모 클래스의 생성자를 불러와 초기화하는 것입니다.



```
class BTS(KPopGroup): # 괄호 안의 KPopGroup을 상속 받는다는 뜻입니다.
    def __init__(self):
        super().__init__("BTS") # super() : 부모 클래스의 생성자

    def greet(self): # 메서드 오버라이딩 (부모 클래스의 함수를 덮어쓰기)
        print(f"(둘, 셋) 방! 탄! 안녕하세요, {self.name}입니다!")
        # 잠깐, BTS 클래스에는 self.name이 없는데요? 그것도 다 물려받을까요?
```

클래스의 상속

- 부모 클래스에 있던 greet 메서드(함수)가 재정의(덮어쓰기)된 것에 주목해봐요!
- greet 함수 내에서 self.name이 갑자기 튀어나오네요. 이것도 상속을 받은 걸까요?

```
class BTS(KPopGroup): # 괄호 안의 KPopGroup을 상속 받는다는 뜻입니다.
    def __init__(self):
        super().__init__("BTS") # super() : 부모 클래스의 생성자

    def greet(self): # 메서드 오버라이딩 (부모 클래스의 함수를 덮어쓰기)
        print(f"(둘, 셋) 방! 탄! 안녕하세요, {self.name}입니다!")
        # 잠깐, BTS 클래스에는 self.name이 없는데요? 그것도 다 물려받을까요?
```

클래스의 상속

- greet 메서드를 호출해볼까요?
- 잘 보니, 부모 클래스로부터 self.name같은 멤버 변수들을 다 물려받되, 새로 '재정의'가 이뤄진 것은 자식 클래스의 것이 우선적으로 실행됩니다.
- 참고: 이것을 유식한 말로 **메서드 오버라이딩**(method overriding)이라고 합니다.
- 단어가 간지나죠? 여러분도 잘 써먹으시기 바랍니다.

```
1  bts = BTS()  
2  
3  # 자식 클래스에서 덮어쓰기된 메서드  
4  bts.greet()
```

(둘, 셋) 방! 탄! 안녕하세요, BTS입니다!

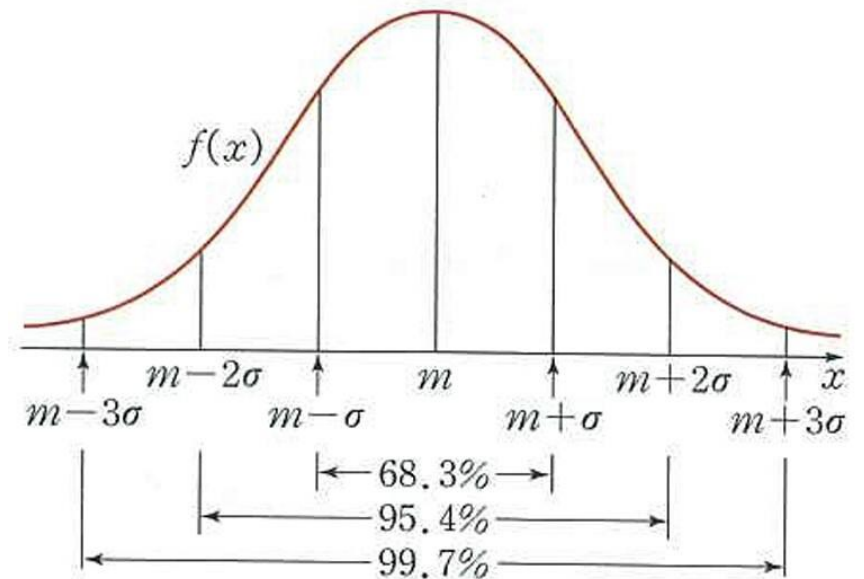
계산기 클래스 상속해보기

- 통계학자들이 좋아하는 정규분포로부터 확률을 구하려면 적분을 계산해야 합니다.
- 이미 다 만든 적분 클래스를 수정하지 말고, 상속을 해봅시다.
- 참고: 정규분포의 확률밀도함수 식이 그리 간단하지 않으니, $\mu(\mu)$ 와 양수 $\sigma(\sigma > 0)$ 에만 의존함을 기억해주시면 됩니다.

$$f(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{(x - \mu)^2}{2\sigma^2}\right)$$

Diagram illustrating the components of the normal distribution formula:

- ②: $\frac{1}{\sigma\sqrt{2\pi}}$ (Coefficient)
- ①: \exp (Exponential function)
- ③: $-\frac{(x - \mu)^2}{2\sigma^2}$ (Exponent)



계산기 클래스 상속해보기

- 상속 예시 (누적 확률에서 근사가 쓰였습니다.)
- 참고: 클래스의 멤버 변수(self.mu, self.sigma)가 마치 **전역 변수**(global variable)의 느낌입니다. 반대말은 **지역 변수**(local variable)였습니다.

```
# 정규분포 확률 계산기
import numpy as np

class NormalDistribution(Integral): # 아까 전 적분 클래스를 상속받음(확장함)
    def __init__(self, mu=0, sigma=1):
        # 정규분포 함수를 적분 클래스(부모 클래스)의 생성자에 넣음 (적분해서 확률 구하기!)
        super().__init__(lambda x: 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-(x-mu)**2/(2*sigma**2)))
        self.mu = mu
        self.sigma = sigma

    def cumul_prob(self, x):
        return self.integral(self.mu - 1000 * self.sigma, x)

    def prob(self, end, start=0):
        # return self.cumul_prob(end) - self.cumul_prob(start)
        return self.integral(start, end)
```


계산기 클래스 상속해보기

- 참고: 당연하게도, 클래스의 메서드 내부에서 다른 메서드를 호출해도 됩니다.
- 순서가 뒤죽박죽이어도 됩니다. 이는 그냥 파이썬 파일에서도 마찬가지입니다.

```
class NormalDistribution(Integral):  
    (...)  
  
    def prob(self, end, start=0):  
        return self.integral(start, end)  
  
    def cumul_prob(self, x):  
        return prob(start=self.mu-1000*self.sigma, end=x)
```

```
class NormalDistribution(Integral):  
    (...)  
  
    def cumul_prob(self, x):  
        return self.integral(self.mu - 1000 * self.sigma, x)  
  
    def prob(self, end, start=0):  
        return self.cumul_prob(end) - self.cumul_prob(start)
```

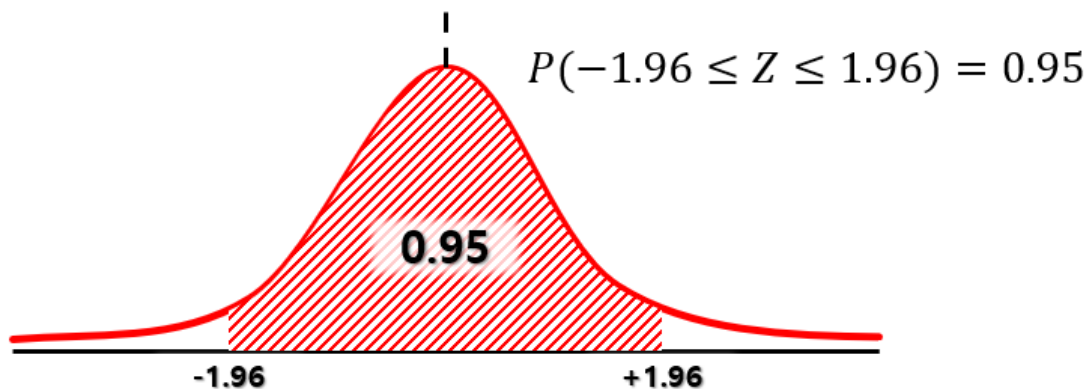
계산기 클래스 상속해보기

- 참고: 새로운 걸 만들었으면 여러 가지 테스트를 진행해 봐야겠죠.
- 예를 들면, 교과서에 계속 나오는 1.96같은 숫자를 집어넣어보면 어떻게 될까요?
- 직접 코드를 통해 이것저것 변형해서 실행해 보세요.

(단, Z 가 표준정규분포를 따르는 확률변수일 때,

$P(|Z| \leq 1.96) = 0.95$ 로 계산한다.) [3점]

① 15.2 ② 15.4 ③ 15.6 ④ 15.8 ⑤ 16.0



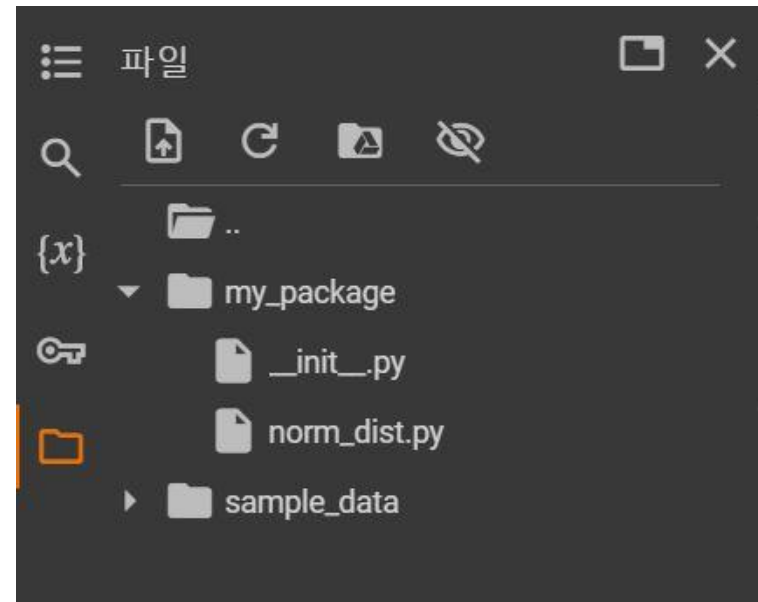
```
1  nd = NormalDistribution()  
2  
3  nd.prob(start=-1.96, end=1.96)  
  
0.9500042067700077
```

클래스를 모듈로 만들기

- 아까 적분 계산기와 정규분포 확률 계산기가 언제 한 번 필요할 때가 있겠죠?
- 그것을 모듈로 만들어서 우리가 직접 불러오겠습니다.
- 패키지로 잘 만들면, 우리가 쓴 코드를 `import`문으로 불러올 수 있습니다!
- 참고: 모듈은 함수, 변수, 클래스를 모아놓은 .py 파일을 말합니다.
- 참고: 모듈이 모여면 패키지가 되고, 패키지가 모여면 라이브러리가 됩니다.

클래스를 모듈로 만들기

- 저는 늘 실습 코드를 Colab에서 적는데, 왼쪽 폴더에서 my_package 폴더를 만들게요.
- 그리고 적당히 파이썬 파일을 만들어서 norm_dist.py로 만들겠습니다.
- 마지막으로, my_package 폴더 안쪽에 빈 파일 `__init__.py`를 생성해주세요. (이것이 있어야 컴퓨터가 우리 파일을 모듈로 판단합니다.)



클래스를 모듈로 만들기

```
# norm_dist.py

import numpy as np

class Integral:
    """
    적분 계산을 위한 클래스입니다. f는 피적분 함수입니다.
    """
    def __init__(self, f):
        self.f = f # integrand

    def set_integrand(self, f):
        self.f = f

    def integral(self, a, b, N=10000):
        """
        적분 구간을 아주 잘게 쪼개서 컴퓨터의 힘으로 근사적으로 적분을 계산 (리만 합의 극한)
        """
        h = (b-a) / N # 적분 구간을 N = 10000등분하여 계산하되
        if h > 0.01: # 적분 구간이 너무 넓을 시 소구간 길이를 작은 상수 0.01로 강제하기
            h = 0.01
            N = int((b-a) / h)
        P = [a + i*h for i in range(N)]
        riemann_sum = 0
```

클래스를 모듈로 만들기

```
    for i in range(N):
        x_i = P[i]
        riemann_sum += h * self.f(x_i) # 적분 근사 계산
    return riemann_sum

class ModularisedNormalDistribution(Integral): # 아까 전 적분 클래스를 상속받음(확장함)
    """
    정규분포 확률 계산기입니다.
    mu=0, sigma=1이 기본값이며, sigma는 양수에서만 작동합니다.
    """
    def __init__(self, mu=0, sigma=1):
        # 정규분포 함수를 적분 클래스(부모 클래스)의 생성자에 넣음 (적분해서 확률 구하기!)
        super().__init__(lambda x: 1/(np.sqrt(2*np.pi)*sigma) * np.exp(-(x-mu)**2/(2*sigma**2)))
        self.mu = mu
        self.sigma = sigma

    def cumul_prob(self, x):
        return self.integral(self.mu - 1000 * self.sigma, x)

    def prob(self, end, start=0):
        # return self.cumul_prob(end) - self.cumul_prob(start)
        return self.integral(start, end)

if __name__ == "__main__": # 이 파일이 .py 파일로 실행되면 아래를 실행하고, 그렇지 않으면 실행하지 않음
    mnd = ModularisedNormalDistribution() # 이름 바꿈에 주목!
    print(mnd.prob(start=-1.0, end=1.0))
```

클래스를 모듈로 만들기

- 그러면 우리가 만든 패키지와 모듈을 `import`할 수 있습니다. 멋있죠?
- 클래스에는 변수, 함수를 다 담을 수 있고, 재사용성도 좋아서, 내가 만든 기능을 모듈로 만들 때 **클래스로 만들어서** 모듈에 넣으면 유용합니다.
- Colab으로 실행 시, 클래스의 이름을 새로 바꾸었음에 유의 바랍니다.

```
1 from my_package import norm_dist
2
3 mnd = norm_dist.ModularisedNormalDistribution() # 모듈.클래스()인 것에 주목!
4 mnd.prob(start=-1.96, end=1.96)
```

0.9500042067700077

```
1 # import문을 조금 다르게 쓸 수도 있습니다.
2 from my_package.norm_dist import ModularisedNormalDistribution
3
4 mnd = ModularisedNormalDistribution()
5 mnd.prob(start=-1.96, end=1.96)
```

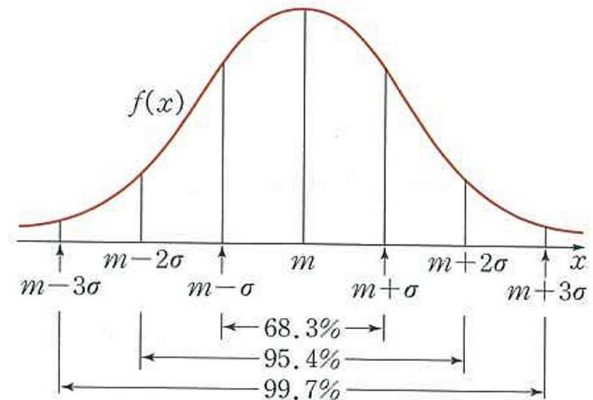
0.9500042067700077

If `__name__ == "__main__":`

- 근데, 아까 모듈 파일 밑에 이런 게 있었죠?

```
if __name__ == "__main__": # 이 파일이 .py 파일로 실행되면 아래를 실행하고, 그렇지 않으면 실행하지 않음
    mnd = ModularisedNormalDistribution() # 이름 바뀜에 주목!
    print(mnd.prob(start=-1.0, end=1.0))
```

- 모듈로 불러올 때에는 이 내용이 전혀 실행되지 않았는데요,
- 이 파일을 모듈이 아니라 직접 실행할 때는 저 값이 print가 될까요?
- 질문: 위의 결과를 예측해보세요.
- 힌트: 아까 그림…!

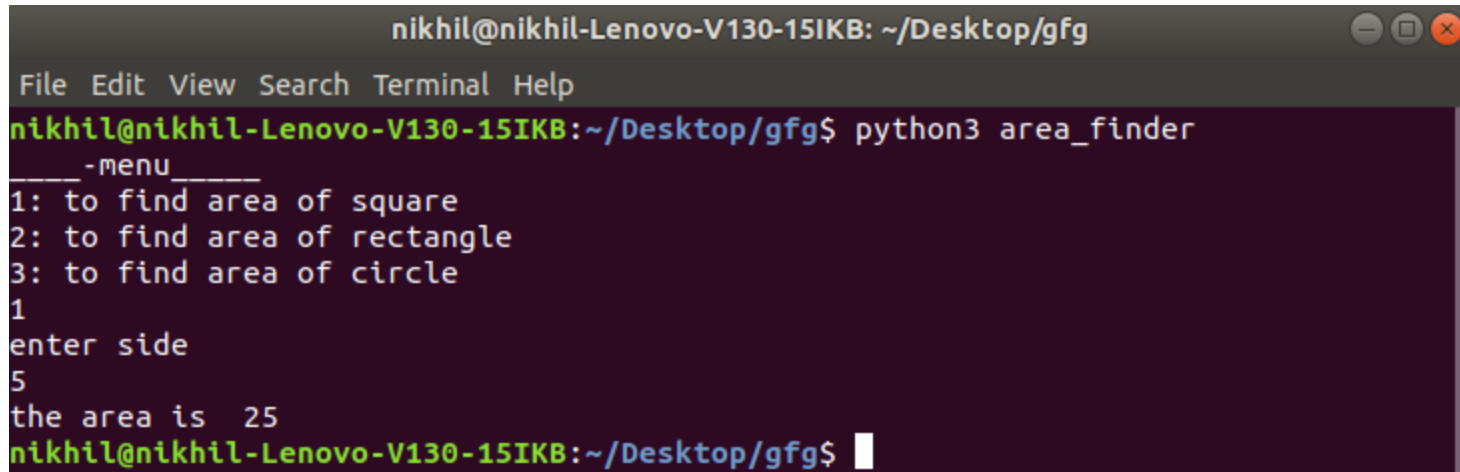


If `__name__ == "__main__":`:

- 이제 Colab에다가 이렇게 적어봅시다.
- 맨 앞 느낌표는 주피터 노트북에서 셀 명령어를 실행하게 해줍니다.

`!python ./my_package/norm_dist.py`

- 이 간단한 명령어를 가장 마지막에서야 알려드리는군요...
- 파이썬 파일로 실행하면 `if __name__ == "__main__":`의 아래 내용이 실행됩니다.



```
nikhil@nikhil-Lenovo-V130-15IKB: ~/Desktop/gfg
File Edit View Search Terminal Help
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$ python3 area_finder
_____
- menu
1: to find area of square
2: to find area of rectangle
3: to find area of circle
1
enter side
5
the area is 25
nikhil@nikhil-Lenovo-V130-15IKB:~/Desktop/gfg$
```

If `__name__ == "__main__":`

- 모듈 내에서 사용하는 테스트 코드를 추가하고 싶을 때 이를 사용하면 효율적입니다.
- 그 테스트 내용은 우리 마음대로 (당연히) 바꿀 수 있습니다.
- 아래에서 바로 보이진 않으나 내부적으로 전부 클래스로 동작하도록 만들어졌습니다.
- 역시, **폼은 일시적이어도 ‘클래스’는 영원합니다!**

```
1 !python ./my_package/norm_dist.py
2 # 파이썬 파일로 실행하면 if __name__ == "__main__": 이라고 적힌 것 아래가 실행됩니다.
```

```
0.6826894905239508
```

이 질문에 답을 찾아보세요

- 이쯤에서 2주차 “파이썬 조감도”를 다시 한 번 보시는 건 어떠신가요?

마지막으로, 아쉬운 점

- pandas나 matplotlib, opencv 등의 외부 라이브러리를 커버하지 못해서 아쉽습니다.
- 파이썬의 강력한 부분 중 하나는 라이브러리인데, 매우 죄송하게 되었습니다.
- with문을 이용한 파일 입출력이나 예외 처리를 다루지 못한 것도 아쉽습니다.
- 지금까지의 내용만 알면 이 주제들을 바로 공부할 수 있습니다.
- 인터넷에 관련 자료(위키독스 등)가 많습니다. 궁금하신 분은,
- 이렇게 말하기 참 무책임한 걸 알지만, 구글링을 해봅시다...

```
pip install rembg
```

Remove Image Background using Python

```
from rembg import remove
from PIL import Image
input_path = 'cl.jpg'
output_path = 'output.png'
input = Image.open(input_path)
output = remove(input)
output.save(output_path)
```

#clcoding.com



마지막으로, 부탁드립니다 점

- 그래도 파이썬의 강력함을 많이 보여드린 것 같아 다행입니다! (소기 목적)
- 파이썬에 거부감을 없애고 친숙함을 쌓아드렸는지는 솔직히 자신이 없네요...☹
- 지금 당장 디테일한 내용을 까먹더라도 나중에 스스로 계속 다시 공부해보세요!
- 모든 기록은 깃허브 레포지토리에 있으니, 하나씩 머릿속 로컬 저장소에 넣어보세요.



우리 스터디에서 다룰 주요 주제

- 파이썬 조감도
- 변수와 리스트
- 문자열, 튜플, 딕셔너리
- 조건문과 반복문
- 함수와 모듈
- 클래스와 상속
- 파이썬을 이용한 문제 해결 (간단한 알고리즘 문제들)
- 파이썬 라이브러리로 할 수 있는 일 (이미지 처리, 엑셀 데이터 처리, 그래프 그리기, ...)

여기까지, DEVELOP이었습니다!

- 다음에 만나요!
- 2024년 6월 4일
- 김현규 드림

