OOP와 의견들

소요시간 : 7 min

> OOP에 대한 기본개념을 알아봅니다.

OOP에 대한 기본개념을 숙지한다.

💡 생각하는 시간 : OOP(**Object Oriented Programming**)

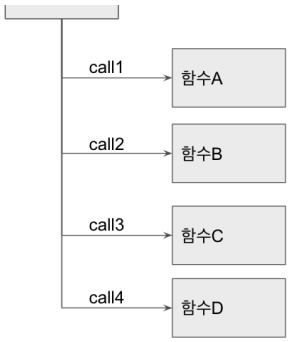
- 쉽게 생각해서 세상에 있는 실체가 있는 모든 물체를 클래스와 인스턴스, 함수, 변수라는 object로 변화시켜서 프로그램을 구성한다고 생각하자.
- OOP의 기본 전제는 기능(함수, 변수) **재사용이 가능하도록 설계 및 프로그래밍** 했는지다.
- 프로그래밍에서 사용되는 대부분의 개념은 최소비용으로 최대효율을 얻기위해 개발되었다. OOP도 그런 개념의 일부일 뿐이다.

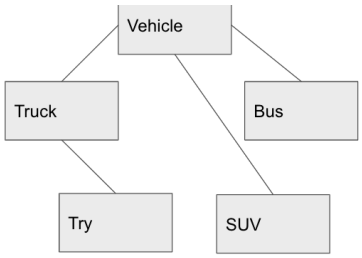
```
# 인트로 코딩

# 난수 다루기(1~100 사이 정수형 난수를 구하자.)
# 정수를 입력받고, 입력숫자가 출력될 때까지 입력ᄌ

import random

input_num = int(input())

for i in range(input_num):
  ran_num = random.randint(1,100)
  print(ran_num, end = ' ')

  if ran_num == input_num:
    print("input random number print")
    continue

else:
  print('exit')
```

OOP에 대한 의견

- 중요한 것은 용어보다는, 현실에서 발생할 수 있는 **특정 object를 컴퓨터라는 도구에 인식시키는 것**이라 할 수 있다.
- 대부분의 분야에서 OOP의 개념을 적용하여 프로그래밍을 수행한다.
- 기본개념 : **설계(사람이 이해하는 방식)와 구현할 소스코드(컴퓨터가 이해하는 방식)** 간의 상호이해가 중요하다.
    - HW&SW 성능증가(CPU성능 증가, 소프트웨어 다중실행) 덕분에 OOP의 모든 기능을 활용할 필요는 없다.
        - OOP의 개념을 무분별하게 활용하면 **유지보수**가 어려워질 수도 있기때문에 설계방향 및 서비스기능에 따라 사용해야 한다.
- OOP의 어려운 점 :
    - OOP는 하나의 패러다임일 뿐이기 때문에 기존의 프로그래밍 패러다임(Procedural Programming, Functional Programming, ...)들과 우열을 가릴 필요는 없다.
    - OOP는 주관성이 높으므로, 보편적으로 활용되는 개념에 대해 배운다.(**주관성이 높다는 뜻은 소프트웨어 서비스 설계방향에 영향을 많이 받는다.**)
    - OOP를 제대로 하는 법은 프로그래밍뿐만 아니라 다양한 도메인에서 **재사용 가능한 클래스, 메소드(기능) 설계**가 중요하다.

OOP 이전에는 어떻게 했을까?



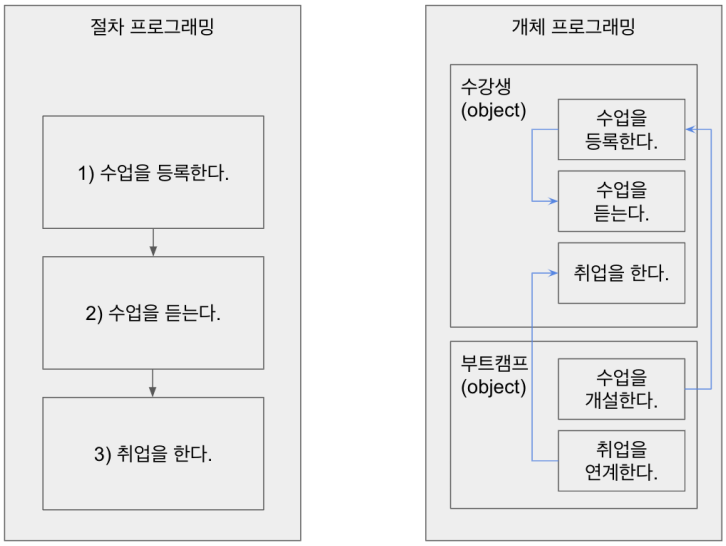


- OOP 개념이 나오기 전에는, 배열과 함수, 변수를 많이 생성하고 활용하여 **최대한 많은 기능을 적은 양의 소스코드파일에 담았다.**
- 속성과 기능이 증가할 때마다 **배열과 함수를 계속 생성**해야했기 때문에 소스코드를 관리하는데 있어서 비효율이 발견되었다.
    - 이에 따라 **속성과 기능을 object라는 최소단위로 분리**하는 OOP의 개념이 나오기 시작했다.
    - 프로그래밍에서 OOP의 개념을 항상 활용하지는 않는다.

      (데이터 분석을 진행하는 경우, 모듈과 라이브러리를 활용한 분석 인사이트가 중요하다.)

OOP의 필요성

- data-driven(**데이터기반 의사결정**), **컴퓨터하드웨어성능, 데이터양 증가**에 따라 OOP활용도 증가하였다.
    - 프로그래밍 패러다임 : OOP와 Procedural Programming, functional Programming에 대해 알아본다.
    - 함수형은 함수의 사용을 극대화시켜서 **코드의 가독성**을 높여주는 형태이다. 프로그래밍 코드를 특정 상황에 특정 함수를 사용하기위해 고안된 방법이다.



OOP와 일상생활

- 일상 생활에서 볼 수 있는 것, **실제로 머릿속에서 떠올릴 수 있는 것**을 프로그래밍하는 것이 OOP의 중요한 점이다.
- 기능별로 개체가 효율적으로(**재사용가능하도록**) 분리되어야 한다. 그래서 설계가 중요하다.
- 위의 두 개의 프로그래밍에 대한 내용을 파이썬 코드로 작성한다.

```
# 케이스 1 : 함수활용
def carAttribute():
  speed = 50
  color = 'black'
  model = 'CarModel'
  return speed,color,model


# 케이스 2 : 변수활용
# speed = 50
# color = 'black'
# model = 'CarModel'

print("Car Procedural Create Complete")
#print("Car Attribute ", speed,color,model) 
print("Car Attribute ", carAttribute()) # re
```

```
# 아래와 같이 클래스 선언 순서에 관계없이 실행된다
# 절차 프로그래밍과 다르게 기능별로 수행되기 때문
class Bus:
  def __init__(self, speed, color):
      self.speed = speed
      self.color = color

  def drive_bus(self):
    self.speed = 70

class Car:
  def __init__(self, speed, color, model):
    self.speed = speed
    self.color = color
    self.model = model

  def drive(self):
    self.speed = 50



myCar = Car(0,"green", "testCar")
myBus = Bus(0,'black')
print("Car Object Create Complete")
print("Car Speed ", myCar.speed)
print("Car Color ", myCar.color)
print("Car Model ", myCar.model)
print("Bus color ", myBus.color)

#Car object method Call
myCar.drive()
myBus.drive_bus()
print("Car Speed by drive ", myCar.speed) # '
print("Bus Speed by drive ", myBus.speed)
```

Mark as Completed

< Prev Next >