AI Programming

Lecture 19

Assignment 14 Solution

```
class BankAccount():

    def __init__(self, name, pin, balance=10000):
        self.__name = name
        self.__pin = pin
        self.__balance = balance

    def getPIN(self):
        return self.__pin

    def getBalance(self):
        return self.__balance

    def setBalance(self, balance):
        self.__balance = balance

    def checkPin(self, pin):
        return self.getPIN() == pin

    def deposit(self, amount):
        self.setBalance(self.getBalance() + amount)
        print("ATM) 입금 후 잔액은 %d원 입니다." % self.getBalance())

    def withdraw(self, amount):
        if self.getBalance() >= amount:
            self.setBalance(self.getBalance() - amount)
            print("ATM) 출금 후 잔액은 %d원 입니다." % self.getBalance())
        else:
            print("ATM) 잔액이 부족합니다.")
```

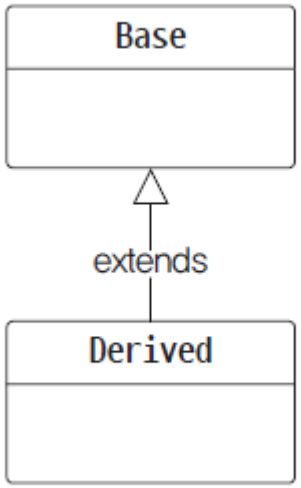
Assignment 15 Solution

```python
class BankAccount:

    accounts = 0

    def __init__(self, name, balance):
        self.name = name
        self.balance = balance
        BankAccount.accounts += 1

    def deposit(self, amount):
        self.balance += amount

    def withdraw(self, amount):
        if self.balance >= amount:
            self.balance -= amount

    def __repr__(self):
        return f"{self.name} 계좌 잔액 {int(self.balance)}"

    def __del__(self):
        print(f"{self.name} 계좌 해지")
        BankAccount.accounts -= 1
```

```python
class SavingsAccount(BankAccount) :
    def __init__(self,  name, balance,  interest_rate):
        super().__init__(name, balance)
        self.interest_rate =interest_rate

    def deposit(self, amount):
        super().deposit(amount + amount * self.interest_rate)


class CheckingAccount(BankAccount) :
    def __init__(self, name, balance, withdraw_charge):
        super().__init__(name, balance)
        self.withdraw_charge = withdraw_charge

    def withdraw(self, amount):
        super().withdraw(amount + self.withdraw_charge)
```

Inheritance

- **is-a relationship (상속 O)**
  - 승용차는 차량의 일종이다 (Car is a Vehicle)
  - 강아지는 동물의 일종이다 (Dog is an Animal)
  - 원은 도형의 일종이다 (Circle is a Shape)
- **has-a relationship (상속 X)**
  - 도서관은 책을 가지고 있다 (Library has a Book)
  - 거실은 소파를 가지고 있다 (Living room has a Sofa)



Inheritance

- **is-a relationships**
  - Vehicle, Car, Truck
  - Employee and Manager
  - Shape, Circle, Rectangle
- **has-a relationships**
  - Card and Deck
  - Department, Course, Student

Is-A

- **Vehicle, Car, and Truck**
  - class Vehicle
    - name
    - __init__, drive (abstract), stop (abstract)
  - class Car(Vehicle)
    - drive (overriding), stop (overriding)
  - class Truck(Vehicle)
    - drive (overriding), stop (overriding)

Is-A

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

    def drive(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")

    def stop(self):
        raise NotImplementedError("이것은 추상메소드입니다. ")
```

```
class Car(Vehicle):
    def drive(self):
        return '승용자를 운전합니다. '

    def stop(self):
        return '승용자를 정지합니다. '


class Truck(Vehicle):
    def drive(self):
        return '트럭을 운전합니다. '

    def stop(self):
        return '트럭을 정지합니다. '


cars = [Truck('truck1'), Truck('truck2'), Car('car1')]

for car in cars:
    print(car.name + ': ' + car.drive())
```

Is-A

- **Employee and Manager**
  - class Employee
    - 속성변수: name, salary
    - Methods: __init__, getSalary (접근자)
  - class Manager(Employee)
    - 속성변수: name, salary, bonus
    - Methods: __init__, getSalary, __repr__

Is-A

```python
class Employee:
    def __init__(self, name, salary):
        self.name = name
        self.salary = salary

    def getSalary(self):
        return salary

class Manager(Employee):
    def __init__(self, name, salary, bonus):
        super().__init__(name, salary)
        self.bonus = bonus

    def getSalary(self):
        salary = super().getSalary()
        return salary + self.bonus

    def __repr__(self):
        return f"이름: {self.name}, 월급: {self.salary}, 보너스: {self.bonus}"

kim = Manager("김철수", 20000, 1000)
print(kim)
```

Is-A

- **Shape, Circle, Rectangle**
  - class Shape
    - name
    - getArea (abstract method)
  - class Circle(Shape)
    - name, radius
    - getArea (overriding)
  - class Rectangle(Shape)
    - name, width, height
    - getArea (overriding)

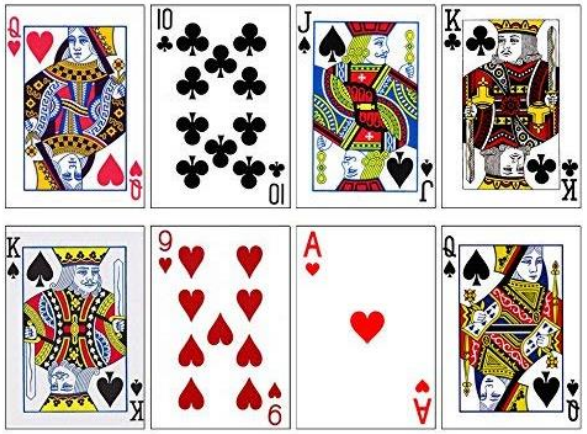
Is-A

```python
import math

class Shape:
    def __init__(self, name):
        self.name = name
    def getArea(self):
        raise NotImplementedError("Abstract method")

class Circle(Shape):
    def __init__(self, name, radius):
        super().__init__(name)
        self.radius = radius

    def getArea(self):
        return  math.pi*self.radius**2
```

```python
class Rectangle(Shape):
    def __init__(self, name, width, height):
        super().__init__(name)
        self.width = width
        self.height = height

    def getArea(self):
        return  self.width*self.height

shapeList = [ Circle("c1", 10), Rectangle("r1", 10, 10) ]
for s in shapeList:
    print(s.getArea())
```

Has-A

- **Card and Deck**
  - class Card
    - suit, rank
    - __init__, __str__
  - class Deck
    - cards
    - __init__, __str__



Has-A

```
class Card:
    suitNames = ['S', 'D', 'H', 'C']
    rankNames = ['A', '2', '3', '4', '5', '6', '7',
                 '8', '9', '10', 'J', 'Q', 'K']

    def __init__(self, suit, rank):
        self.suit = suit
        self.rank = rank

    def __str__(self):
        return Card.suitNames[self.suit]+" "+Card.rankNames[self.rank]
```

```
class Deck:
    def __init__(self):
        self.cards = []

        for suit in range(4):
            for rank in range(13):
                card = Card(suit, rank)
                self.cards.append(card)

    def __str__(self):
        lst = [str(card) for card in self.cards]
        return str(lst)

deck = Deck()
print(deck)
```

Has-A

- **Department, Course, Student**
  - class Department
    - name, courses
    - __init__, addCourse, __str__
  - class Course
    - name, students, department
    - __init__, addStudent, __str__
  - class Student
    - name, id
    - __init__, __str__

Has-A

```python
class Department:
    def __init__(self, name):
        self.name = name
        self.courses = []

    def addCourse(self, name):
        course = Course(name, self)
        self.courses.append(course)
        return course

    def __str__(self):
        outStr = f"{self.name}학과₩n"

        for course in self.courses:
            outStr += f"₩t{course}₩n"

            for student in course.students:
                outStr += f"₩t₩t{student}₩n"

        return outStr
```

```python
class Course:
    def __init__(self, name, department):
        self.name = name
        self.students = []
        self.department = department

    def addStudent(self, name, id):
        student = Student(name, id)
        self.students.append(student)
        return student

    def __str__(self):
        return f"{self.name}: {len(self.students)}명 수강"

class Student:
    def __init__(self, name, id):
        self.name = name
        self.id = id

    def __str__(self):
        return f"{self.id} 학번 {self.name}"
```

Has-A

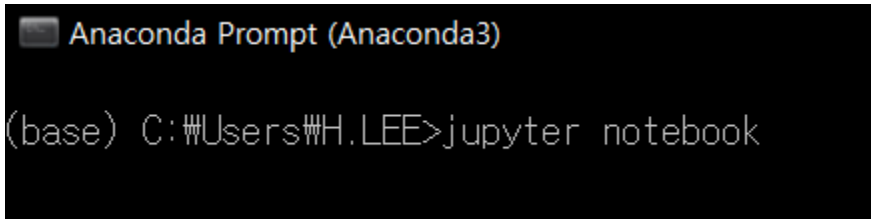
```
dept = Department("정보통신공")

cor1 = dept.addCourse("자료구조")
cor2 = dept.addCourse("AI프로그래밍")

std1 = cor1.addStudent("Kim", 202101)
std2 = cor1.addStudent("Lee", 202102)
std3 = cor2.addStudent("Park", 201901)
std4 = cor2.addStudent("Choi", 201801)
std5 = cor2.addStudent("Lim", 202103)

print(dept)
```

```
정보통신공학과
        자료구조: 2명 수강
                202101 학번 Kim
                202102 학번 Lee
        AI프로그래밍: 3명 수강
                201901 학번 Park
                201801 학번 Choi
                202103 학번 Lim
```

Jupyter Notebook

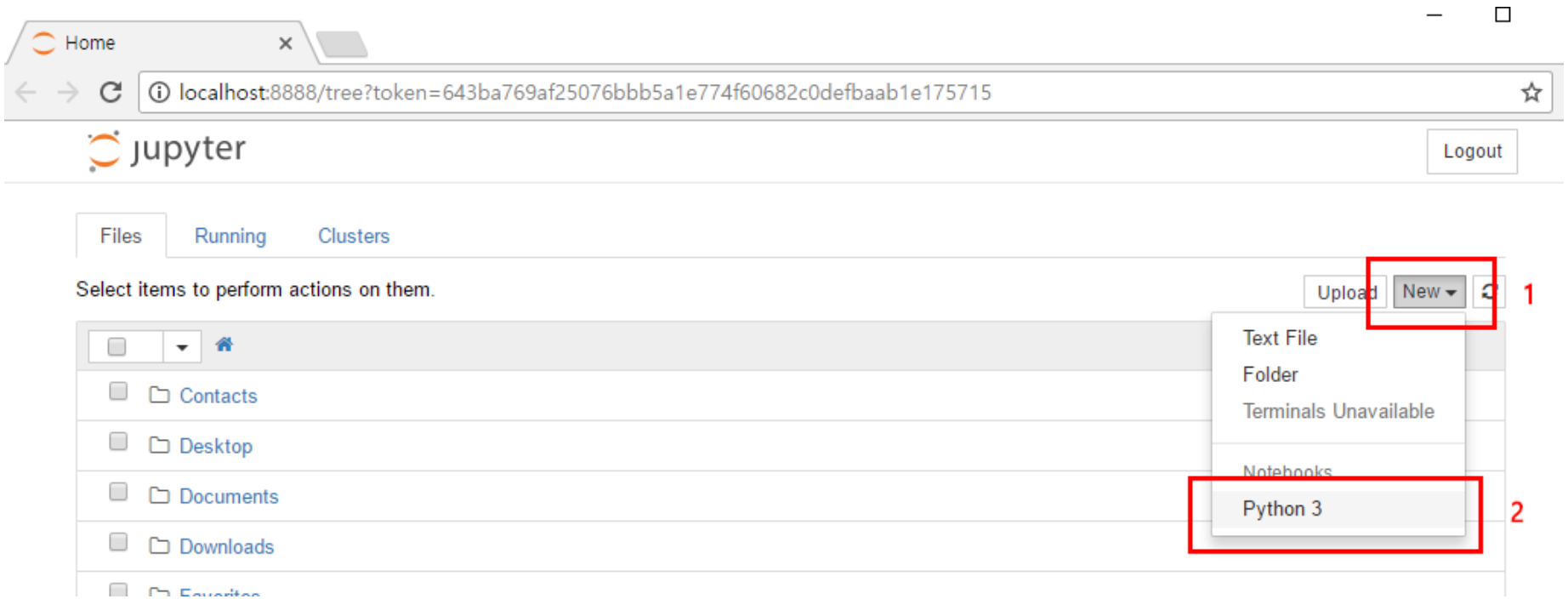
Jupyter Notebook

- **실행**
  - 찾기 → Jupyter notebook
  - Anaconda → jupyter notebook



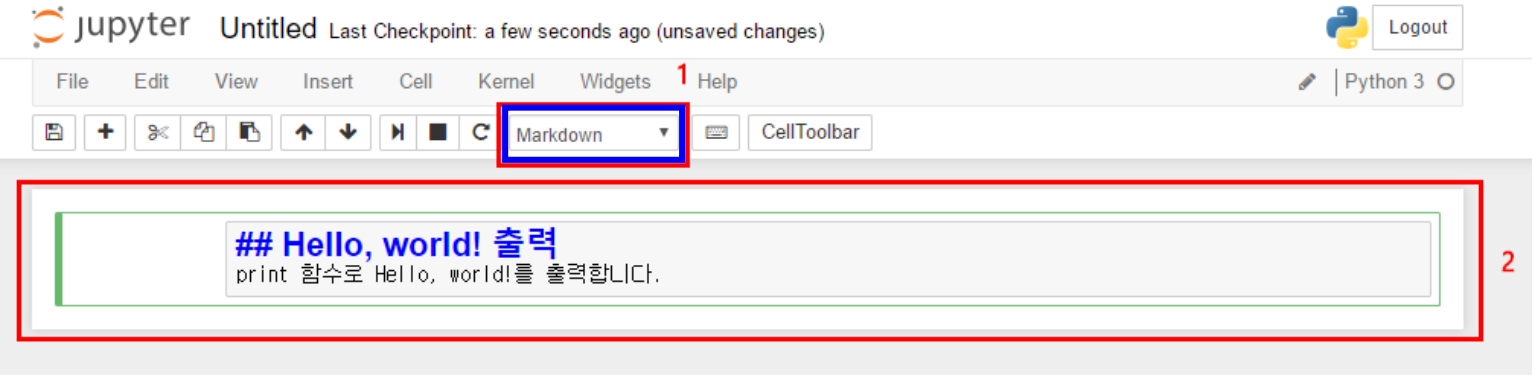
Jupyter Notebook

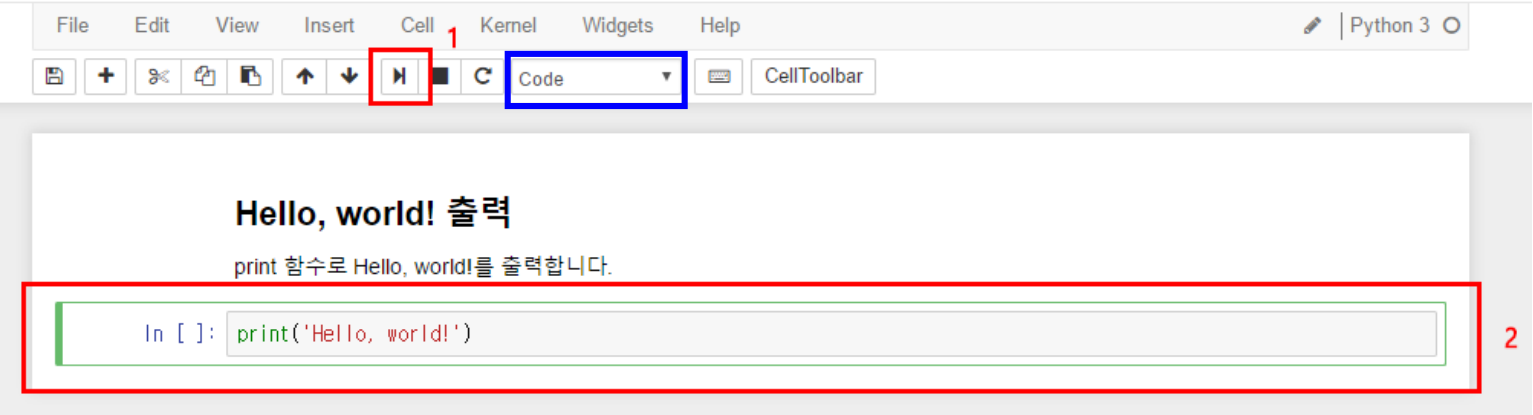
- ipynb 노트북 생성



Jupyter Notebook

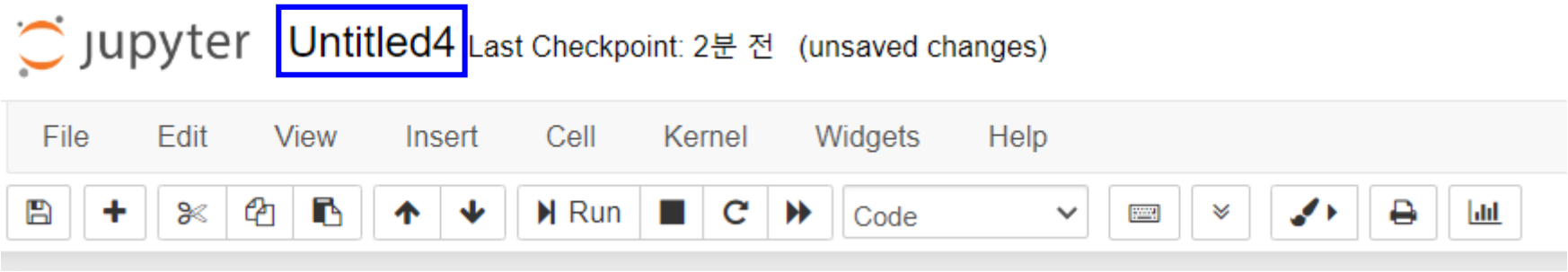
- **입력 모드**
  - Markdown (주석)
    - 셀 선택 → m
  - 코드 입력 모드
    - 셀 선택 → y
    - 코드 실행
      - Ctrl + Enter
      - Shift + Enter





Jupyter Notebook

- **노트북 이름 변경**
  - File → Rename

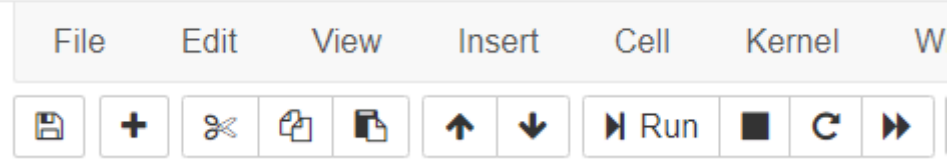


Jupyter Notebook

- **노트북 저장/불러오기**
  - 저장: Ctrl + S
    - 디렉토리: 노트북 생성시 결정
  - 불러오기
    - File → Open
    - ipynb: 읽기 & 편집 & 실행
    - py: 읽기 & 편집만 가능

Jupyter Notebook

- **셀 편집**
  - Edit → Cut Cells, Copy Cells, Delete Cells,…
  - 단축키
    - d+d: 셀 삭제
    - a: 위에 셀 추가
    - b: 아래에 셀 추가
    - Crtl + c, Crtl + v



Jupyter Notebook

- **Markdown**
  - Header (제목)

```
# Header 1 : # 1개 사용
## Header 2 : # 2개 사용
### Header 3 : # 3개 사용
#### Header 4 : # 4개 사용
##### Header 5 : # 5개 사용
###### Header 6 : # 6개 사용
####### Header 7 : # 7개 사용; 6개까지 header로 활용 가능
```

Header 1 : # 1개 사용

Header 2 : # 2개 사용

Header 3 : # 3개 사용

Header 4 : # 4개 사용

Header 5 : # 5개 사용

Header 6 : # 6개 사용

####### Header 7 : # 7개 사용; 6개까지 header로 활용 가능

Jupyter Notebook

- **Markdown**
  - Bullet & Level: 들여쓰기로 표현

```
1. 첫 번째
2. 두 번째
3. 세 번째
```

1. 첫 번째
2. 두 번째
3. 세 번째

```
* 별표 (*)
+ 플러스 (+)
- 마이너스 (-)
```

- 별표 (*)
- 플러스 (+)
- 마이너스 (-)

```
* level1
 * level2
 * level2-2
   * level3
```

- level1
  - level2
  - level2-2
    - level3

Jupyter Notebook

- **Markdown**
  - 이미지 삽입
    - Markdown 모드 셀에 이미지 드래그

```
![cap4.png](attachment:cap4.png)
```

Jupyter Notebook

- 모듈 import 확인

In [3]:
```
import numpy as np
import matplotlib.pyplot as plt
import pandas as pd
import sklearn
```