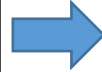


(Ch 17-A) Python Code Reading Recitation B

Functions [1/7]

```
def greet_user(username):  
    """Display a simple greeting."""  
    print("Hello, " + username.title() + "!")  
  
greet_user('jesse')
```



Hello, Jesse!

Username으로 입력 받은 parameter를 앞 문자를 대문자로 바꿔주는 메서드 title을 이용하여 문자열을 출력한다. + 사용시 띄어쓰기를 적용하지 않는다.

```
def describe_pet (pet_name, animal_type = 'dog'):  
    """Display information about a pet."""  
    print("\nI have a " + animal_type + ".")  
    print("My " + animal_type + "'s name is " + pet_name.title() + ".")  
  
# A dog named Willie.  
describe_pet('willie')  
describe_pet(pet_name = 'willie')  
  
# A hamster named Harry.  
describe_pet('harry', 'hamster')  
describe_pet(pet_name = 'harry', animal_type= 'hamster')  
describe_pet(animal_type = 'hamster', pet_name = 'harry')
```



```
I have a dog.  
My dog's name is Willie.  
  
I have a dog.  
My dog's name is Willie.  
  
I have a hamster.  
My hamster's name is Harry.  
  
I have a hamster.  
My hamster's name is Harry.  
  
I have a hamster.  
My hamster's name is Harry.
```

optional parameter인 animal_type에는 default value로 'dog'가 정해져 있다. 따라서, 한 가지 parameter만 입력할 경우 regular parameter인 pet_name으로 값이 들어간다. 반면, 밑 3개의 입력 예시 처럼 parameter 두 개를 입력하거나, 직접 parameter를 지정해주는 경우(직접 입력 시 순서는 상관 없음) animal_type의 value를 바꿀 수 있다. 적절한 문자열에 입력 받은 animal_type, pet_name을 넣어 print함수를 실행해 주게 된다.

Functions [2/7]

```
def get_formatted_name(first_name, last_name, middle_name=""):
    """Return a full name, neatly formatted."""
    if middle_name:
        full_name = first_name + ' ' + middle_name + ' ' + last_name
    else:
        full_name = first_name + ' ' + last_name
    return full_name.title()
```

```
musician = get_formatted_name('jimi', 'hendrix')
print(musician)
musician = get_formatted_name('john', 'hooker', 'lee')
print(musician)
```



Jimi Hendrix
John Lee Hooker

처음의 if, else 구문은 middle_name 파라미터를 입력 받지 않아 Default value인 “가 되었을 때, middle_name을 생략한 풀네임 문자열을 만들고, 따로 입력을 받았을 경우에는 middle_name을 포함한 풀네임 문자열을 만드는 구문이다. get_formatted_name함수는 위의 구문대로 full_name을 만들고, 맨 앞 문자들을 대문자로 바꾼 문자열을 반환하는 함수이다. 위의 musician에는 Jimi Hendrix 아래의 musicia에는 John Lee Hooker 가 저장 되고, print 함수를 통해 각각을 출력해주게 된다. Print 함수는 줄바꿈을 하기 때문에 두 이름은 줄을 바꾸어 출력되게 된다.

Functions [3/7]



```
{'first': 'jimi', 'last': 'hendrix', 'age': 27}
```

```
def build_person (first_name, last_name, age = ''):
    """Return a dictionary of information about a person."""
    person = {'first' : first_name, 'last' : last_name}
    if age:
        person['age'] = age
    return person

musician = build_person('jimi', 'hendrix', age=27)
print(musician)
```

Build_person 함수는 first_name, last_name, age를 입력 받아 person이라는 dictionary를 반환하는 함수이다. Age를 입력받지 않는다면 if 구문 내부로 들어갈 수 없기 때문에 person에 age라는 key 값이 없게 된다. Age를 입력받아야 person에 age라는 key가 생기고, value가 들어가게 된다. Musician에 {'first': 'jimi', 'last': 'hendrix', 'age': 27}이라는 dictionary가 생기고 이를 출력하게 된다.

```
def greet_users(names):
    """Print a simple greeting to each user in the list."""
    for name in names:
        msg = "Hello, " + name.title() + "!"
        print(msg)

usernames = ['hannah', 'ty', 'margot']
greet_users(usernames)
```



```
Hello, Hannah!
Hello, Ty!
Hello, Margot!
```

Greet_usres함수에 usernames라는 리스트를 입력해주게 되면, for 문이 잘 작동하게 된다.(list가 Iterable하기 때문이다) usernames 리스트에서 하나의 문자열씩 꺼내서 name을 바꿔주게 되면, msg가 리스트 속 이름을 하나씩 가져와 hello를 붙여준 문자열로 바뀌고 이를 출력해주게 된다.

Functions [4/7]

```
def print_models(unprinted_designs, completed_models):  
    """
```


```
    Simulate printing each design, until there are none left.  
    Move each design to completed_models after printing.  
    """
```

```
    while unprinted_designs:  
        current_design = unprinted_designs.pop()  
  
        # Simulate creating a 3d print from the design.  
        print("Printing model: " + current_design)  
        completed_models.append(current_design)
```

```
def show_completed_models(completed_models):  
    """Show all the models that were printed."""  
    print("\nThe following models have been printed:")  
    for completed_model in completed_models:  
        print(completed_model)
```

```
unprinted_designs = ['iphone case', 'robot pendant', 'dodecahedron']  
completed_models = []
```

```
print_models(unprinted_designs, completed_models)  
show_completed_models(completed_models)
```



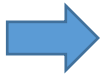
```
Printing model: dodecahedron  
Printing model: robot pendant  
Printing model: iphone case
```

```
The following models have been printed:  
dodecahedron  
robot pendant  
iphone case
```

Print_models함수는 unprinted_designs, completed_models 두 리스트를 입력받아, while 문을 통해 unprinted_designs에서 한 개씩 꺼내 printing model을 한 뒤, completed_models에 append 메서드를 이용해 꺼낸 값을 넣어준다. Show_completed_models함수는 completed_models라는 리스트를 입력받아 for문을 통해 한 개씩 출력해주는 함수이다. Print_models함수를 통해 completed_models로 unprinted_designs를 옮긴 뒤 show_completed_models를 통해 출력해주게 된다.

Functions [5/7]

```
def make_pizza (size, *toppings):  
    """Summarize the pizza we are about to make."""  
    print("\nMaking a " + str(size) + "-inch pizza with the following toppings:")  
    for topping in toppings:  
        print("- " + topping)  
  
make_pizza(16, 'pepperoni')  
make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```

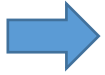


```
Making a 16-inch pizza with the following toppings:  
- pepperoni  
  
Making a 12-inch pizza with the following toppings:  
- mushrooms  
- green peppers  
- extra cheese
```

Make_pizza함수는 size를 정수로 숫자로 입력받고, toppings를 입력 받는다. *는 여러 개의 입력 값을 받음을 의미 한다. 첫 print함수는 피자의 사이즈를 설명해주는 문자열을 출력해주고, size가 숫자이므로 str함수를 통해 문자열로 출력 되도록 한다. Toppings에는 여러 입력값이 들어올 수 있으므로 for 구문을 통해 여러 토핑 들을 출력 할 수 있도록 코딩한 것을 알 수 있다. 맨 밑의 예시의 경우 size = 12, toppings=('mushrooms', 'green peppers', 'extra cheese')이다.

Functions [6/7]

```
def build_profile(first, last, **user_info):  
    """Build a dictionary containing everything we know about a user."""  
    profile = {}  
    profile['first_name'] = first  
    profile['last_name'] = last  
    for key,value in user_info.items():  
        profile[key] = value  
    return profile  
  
user_profile = build_profile('albert', 'einstein', location = 'princeton', field = 'physics')  
  
print(user_profile)
```



```
{'first_name': 'albert', 'last_name': 'einstein', 'location': 'princeton', 'field': 'physics'}
```

**는 parameter=value의 형태로 optional parameter를 입력 받겠다는 의미이다. Build_profile함수는 first,last,user_info를 입력 받아 profile이라는 dictionary에 정리를 해 반환하는 함수이다. Key first_name에는 first를, last_name에는 last를 user_info에서 key로 입력 받은 것은 Key로 value로 입력 받은 것은 value로 dictionary를 만든다. 맨 밑의 예시에서 location에는 princeton이 value로 저장되는 것을 알 수 있다. 반환된 dictionary는 user_profile에 저장되어 print를 통해 출력 된다.

Functions [7/7]

```
def make_pizza(size, *toppings):  
    """Summarize the pizza we are about to make."""  
    print("\nMaking a " + str(size) +  
          "-inch pizza with the following toppings:")  
    for topping in toppings:  
        print("- " + topping)  
  
import pizza as p  
  
p.make_pizza(16, 'pepperoni')  
p.make_pizza(12, 'mushrooms', 'green peppers', 'extra cheese')
```



```
Making a 16-inch pizza with the following toppings:  
- pepperoni  
  
Making a 12-inch pizza with the following toppings:  
- mushrooms  
- green peppers  
- extra cheese
```

5번째 function과 똑같은 함수를 pizza라는 py파일에 저장한뒤 import를 통해 pizza.py라는 파일을 p로 불러냄을 알 수 있다. P.make_pizza를 통해 pizza.py에 있는 make_pizza함수를 불러내는 모습이다. 출력 결과는 당연히 5번째 function과 똑같은 모습이다.

Code with “Dog” Class

```
class Dog():
    """A simple attempt to model a dog."""

    def __init__(self, name, age):
        """Initialize name and age attributes."""
        self.name = name
        self.age = age

    def sit(self):
        """Simulate a dot sitting in response to a command."""
        print(self.name.title() + " is now sitting.")

    def roll_over(self):
        """Simulate rolling over in response to a command."""
        print(self.name.title() + " rolled over!")

my_dog = Dog('willie', 6)
your_dog = Dog('lucy', 3)

print ("My dog's name is " + my_dog.name.title() + ".")
print ("My dog is " + str(my_dog.age) + " years old.")
my_dog.sit()

print ("\nMy dog's name is " + your_dog.name.title() + ".")
print ("My dog is " + str(your_dog.age) + " years old.")
your_dog.sit()
```



출력 결과 :

My dog's name is Willie.
My dog is 6 years old.
Willie is now sitting.

My dog's name is Lucy.
My dog is 3 years old.
Lucy is now sitting.

Dog 클래스는 개를 간단하게 모델링한 클래스이다. 개의 이름과 나이에 대한 속성이 있으며, 생성자, 개가 앉는 상황을 가정한 sit() 메서드, 그리고 개가 구르는 상황을 가정한 roll_over() 메서드가 있다.

위 코드에서는 Dog 클래스로 my_dog, your_dog 객체를 만든 후 각각 이름과 나이 속성을 출력하고 sit() 메서드를 사용한다. 이때 개의 이름은 title()을 이용해 앞글자를 대문자로 바꾸어준다.

Code with “Car” and “ElectricCar” Class [1/4]

car.py

```
""" A class that can be used to represent a car. """

class Car():
    """A simple attempt to represent a car."""

    def __init__(self, manufacturer, model, year):
        """Initialize attributes to describe a car."""
        self.manufacturer = manufacturer
        self.model = model
        self.year = year
        self.odometer_reading = 0

    def get_descriptive_name(self):
        """Return a neatly formatted descriptive name."""
        long_name = str(self.year) + ' ' + self.manufacturer + ' ' + self.model
        return long_name.title()

    def read_odometer(self):
        """Print a statement showing the car's mileage."""
        print("This car has " + str(self.odometer_reading) + " miles on it.")

    def update_odometer(self, mileage):
        """
        Set the odometer reading to the given value.
        Reject the change if it attempts to roll the odometer back.
        """
        if mileage >= self.odometer_reading:
            self.odometer_reading = mileage
        else:
            print("You can't roll back an odometer!")

    def increment_odometer(self, miles):
        """Add the given amount to the odometer reading."""
        self.odometer_reading += miles
```

electric_car.py

```
"""A set of classes that can be used to represent electric cars."""
from car import Car
class Battery() :
    """A simple attempt to model a battery for an electric car."""
    def __init__ (self, battery_size = 60):
        """Initialize the battery's attributes."""
        self.battery_size = battery_size

    def describe_battery (self):
        """Print a statement describing the battery size."""
        print("This car has a " + str(self.battery_size) + "-kWh battery.")

    def get_range(self):
        """Print a statement about the range this battery provides."""
        if self.battery_size == 60:
            range = 140
        elif self.battery_size == 85:
            range = 185

        message = "This car can go approximately " + str(range)
        message += " miles on a full charge."
        print(message)

class ElectricCar (Car):
    """Models aspects of a car, specific to electric vehicles."""

    def __init__ (self, manufacturer, model, year):
        """
        Initialize attributes of the parent class.
        Then initialize attributes specific to an electric car.
        """
        super().__init__ (manufacturer, model, year)
        self.battery = Battery()
```

```
my_used_car = Car ('subaru', 'outback', 2013)
print(my_used_car.get_descriptive_name())
```

```
my_used_car.update_odometer(23500)
my_used_car.read_odometer()
```

```
my_used_car.increment_odometer(100)
my_used_car.read_odometer()
```

출력 결과 :

2013 Subaru Outback
This car has 23500 miles on it.
This car has 23600 miles on it.

Car 클래스는 자동차를 모델링한 클래스이다. 제조사, 모델명, 연식, 주행거리를 나타내는 속성과 생성자 메서드, 차를 설명해주는 스트링을 리턴하는 메서드, 주행거리를 읽는 메서드, 갱신하는 메서드, 증가시키는 메서드로 이루어져 있다. 이 코드에서는 Car 클래스 타입의 객체 my_used_car를 만들어 2013년형 Subaru Outback 기종으로 초기화시키고, get_descriptive_name()를 이용해 차를 설명한다. 또한, update_odometer()로 주행거리를 23500으로 초기화하고 increment_odometer()로 100만큼 증가시킨 후 각각 read_odometer()로 출력시켜준다.

```
my_tesla = ElectricCar ('tesla', 'model s', 2016)
print(my_tesla.get_descriptive_name())
my_tesla.battery.describe_battery()
```

출력 결과 :

2016 Tesla Model S
This car has a 60-kWh battery.

ElectricCar 클래스는 Car 클래스를 상속하며, Battery() 클래스 타입 객체인 battery를 추가로 가지고 있다. Battery() 클래스는 배터리 사이즈를 속성으로 가지며, 배터리 사이즈를 설명해주는 메서드와 한 번 충전으로 갈 수 있는 거리를 설명해주는 메서드를 가지고 있다.

위 코드에서는 ElectricCar 클래스 타입의 my_tesla 객체를 만들어, 2016년형 Tesla Model S 기종으로 초기화한다. 기종을 설명하는 메서드와 배터리를 설명하는 메서드를 호출하며, 이때 배터리 사이즈는 자동적으로 60으로 초기화된다.

Code with “Car” and “ElectricCar” Class

[4/4]

```
from car import Car

my_new_car = Car('audi', 'a4', 2015)
print(my_new_car.get_descriptive_name())

my_new_car.odometer_reading = 23
my_new_car.read_odometer()
```

출력 결과 :
2015 Audi A4
This car has 23 miles on it.

그 전의 코드와 마찬가지로 car.py 코드에서 Car를 import 하여 Car 클래스 타입의 my_new_car 객체를 생성한다. 차의 정보를 2015년 아우디 A4 기종으로 초기화한 후, 이를 출력하는 메서드를 호출한다. 또한 주행거리 속성을 23으로 바꾸고, 이를 출력해준다.

```
from car import Car
from electric_car import ElectricCar

my_bettle = Car('volkswagen', 'beetle', 2015)
print(my_bettle.get_descriptive_name())

my_tesla = ElectricCar('tesla', 'roadster', 2015)
print(my_tesla.get_descriptive_name())
```

출력 결과 :
2015 Volkswagen Beetle
2015 Tesla Roadster

이 코드에서는 Car와 ElectricCar를 모두 import한다. Car 클래스 타입의 my_bettle 객체를 만들어 그 정보를 출력하는 메서드를 호출하고, ElectricCar 클래스 타입의 my_tesla 객체를 만들어 같은 메서드를 호출한다. 각 객체는 2015년형 폭스바겐 Beetle, 2015년형 테슬라 Roadster로 정보가 초기화되어있다.