웹프로그래밍의 기초

Week5

IF;

Python data structure #2 Set & Dictionary

If…

Booleans

- In programming you often need to know if an expression is True or False. You can evaluate any expression in Python, and get one of two answers, True or False.

```
green_lights = True
red_lights = False

if ( green_lights ):
    print("Green means Go.")
if ( red_lights ):
    print("Red means Stop.")
---------------
Green means Go.
```

Booleans (cont’d)

- Most Values are True
  - Almost any value is evaluated to True if it has some sort of content.
  - Any string is True, except empty strings.
  - Any number is True, except 0.
  - Any list, tuple, set, and dictionary are True, except empty ones.
- Some Values are False
  - In fact, there are not many values that evaluate to False, except empty values, such as (), [], {}, "", the number 0, and the value None. And of course the value False evaluates to False.

```
print(bool("abc"))
print(bool(123))
print(bool(["apple", "cherry", "banana"]))
----------------------
True
True
True
```

```
print(bool(False))
print(bool(None))
print(bool(0))
print(bool(""))
print(bool(()))
print(bool([]))
print(bool({}))
----------------------
False
False
False
False
False
False
False
```

Python Conditions and If statements

- Python supports the usual logical conditions from mathematics:

Equals:	a == b
Not Equals:	a != b
Less than:	a < b
Less than or equal to:	a <= b
Greater than:	a > b
Greater than or equal to:	a >= b

- These conditions can be used in several ways, most commonly in "if statements" and loops.

```
>>> a = 33
>>> print(a)
33
```

```
>>> a == 33
True
>>> a == 34
False
>>> a != 34
True
>>> a < 35
True
>>> a > 22
True
```

If – general idea

```
cars = ['bmw', 'audi', 'toyota', 'hyundai']

for car in cars:
    if car == 'bmw':
        print(car.upper())
    else:
        print(car.title())
----------------------
BMW
Audi
Toyota
Hyundai
>
```

If – availability condition

- The **in** keyword is a logical operator, and is used to check if the stated item exists in the condition.

```
requested_pizza_toppings = ['mushrooms', 'onions', 'pineapple']
if 'mushrooms' in requested_pizza_toppings:
    print("Mushrooms are already placed on your pizza.")
--------------------
Mushrooms are already placed on your pizza.
```

- The **not in** keyword is a logical operator, and is used to check if the stated item does not exist in the condition

```
banned_users = ['badboy', 'darkdog', 'blackandrew']
your_id = 'goodstudent'
if your_id not in banned_users:
    print("You are OK to go.")
--------------------
You are OK to go.
```

If – multiple conditions

- The **and** keyword is a logical operator, and is used to combine conditional statements:

```
a = 200
b = 33
c = 500
if a > b and c > a:
  print("Both conditions are True")
-----------------
Both conditions are True
```

- The or keyword is a logical operator, and is used to combine conditional statements:

```
a = 200
b = 33
c = 500
if a > b or a > c:
  print("At least one of the conditions is True")
-----------------
At least one of the conditions is True
```

If statement

- The basic form of if statement consists of **a single if** and its body.

```
no_pokemon = 20
if no_pokemon > 10:
    print("You\'ve got many Pokemon friends.")
-------------------
You've got many Pokemon friends.
```

- When you want to do something in case the condition fails, then **if-else** should be used.

```
no_pokemon = 9
if no_pokemon > 10:
    print("You\'ve got many Pokemon friends.")
else:
    print("You need more Pokemons.")
-------------------
You need more Pokemons.
```

If statement (Cont'd)

- When there are 3 more possible situations, then if-elif-else statement should be used.

```
no_pokemon = 101
if no_pokemon > 10:
       print("You\'re a true Pokemon master.")
elif no_pokemon > 100:
       print("You\'ve got many Pokemon friends.")
else:
       print("You need more Pokemons.")
-------------------
You're a true Pokemon master.
```

If statement (Cont’d)

- When you need to check all the situations, then **multiple if** statements should be used.

```
no_pokemon = 51
if no_pokemon > 10:
       print("You\'re LV1 Pokemon trainer")
if no_pokemon > 50:
       print("You\'re LV2 Pokemon trainer")
if no_pokemon > 100:
       print("You\'re LV3 Pokemon trainer")
-----------------------
You're LV1 Pokemon trainer
You're LV2 Pokemon trainer
```

If to check List objects

- If statement also allows you to find the specific list item that you may want to check.

```
available_toppings = ['mushroom','olives','green peppers','pepperoni','pineapple','extra cheese']
requested_toppings = ['mushroom','french fries','extra cheese']
added_toppings = []

for requested_topping in requested_toppings:
    if requested_topping in available_toppings:
        added_toppings.append(requested_topping)
        print("Adding " + requested_topping + ".")
    else:
        print("Ooops , we don't have " + requested_topping + ". Skipping " + requested_topping + ".")

print("-----------------------------------")
print("We have finished making your pizza, and it has " + " and ".join(added_toppings) + " on top of it.")
```

```
Adding mushroom.
Ooops , we don't have french fries. Skipping french fries.
Adding extra cheese.
-----------------------------------
We have finished making your pizza, and it has mushroom and extra cheese on top of it.
```

Dictionary

Python Collections (Arrays)

- There are four collection data types in the Python programming language:
    - List is a collection which is ordered and changeable. Allows duplicate members.
    - Tuple is a collection which is ordered and unchangeable. Allows duplicate members.
    - Set is a collection which is unordered, unchangeable*, and unindexed. No duplicate members.
        - Set items are unchangeable, but you can remove and/or add items whenever you like.
    - Dictionary is a collection which is ordered** and changeable. No duplicate members.
        - As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.

Dictionary - introduction

- Dictionaries are used to store data values in key:value pairs.
- A dictionary is a collection which is **ordered, changeable and do not allow duplicates**.
  - When we say that dictionaries are ordered, it means that the items have a defined order, and that order will not change. Unordered means that the items does not have a defined order, you cannot refer to an item by using an index.
  - As of Python version 3.7, dictionaries are ordered. In Python 3.6 and earlier, dictionaries are unordered.
- Dictionaries are written with curly brackets, and have keys and values:

Dictionary – general idea

```
# empty dictionary
my_dict = {}
print(my_dict)
---------------------
{}
```

```
# dictionary with integer keys
my_dict = {1: 'apple', 2: 'ball'}
print(my_dict)
---------------------
{1: 'apple', 2: 'ball'}
```

```
# dictionary with mixed keys
my_dict = {'name': 'John', 1: [2, 4, 3]}
print(my_dict)
---------------------
{'name': 'John', 1: [2, 4, 3]}
```

```
# from sequence having each item as a pair
my_dict = dict([(1,'apple'), (2,'ball')])
print(my_dict)


{1: 'apple', 2: 'ball'}
```

Dictionary – accessing and adding items

- Accessing value via key

```
mycardictionary =  {
  "brand": "KIA",
  "model": "Sorento",
  "year": 2012
}
print(mycardictionary)
print(mycardictionary["brand"])
-----------------------
{'brand': 'KIA', 'model': 'Sorento', 'year': 2012}
KIA
```

- Adding and additional pair of key-value

```
mycardictionary['purcharse_year']=2012
print(mycardictionary)

{'brand': 'KIA', 'model': 'Sorento', 'year': 2012, 'purcharse_year': 2012}
```

Dictionary – changing and deleting items.

- Changing the value of the given key

```
mycardictionary =         {
  "brand": "KIA",
  "model": "Sorento",
  "year": 2012
}
print(mycardictionary)
mycardictionary["model"] = "Niro"
print(mycardictionary)
-----------------------
{'brand': 'KIA', 'model': 'Sorento', 'year': 2012}
{'brand': 'KIA', 'model': 'Niro', 'year': 2012}
>
```

- Deleting a key-value pair

```
mycardictionary =         {
  "brand": "KIA",
  "model": "Sorento",
  "year": 2012
}
print(mycardictionary)
del mycardictionary[“model"]
print(mycardictionary)
-----------------------
{'brand': 'KIA', 'model': 'Sorento', 'year': 2012}
{'brand': 'KIA', 'year': 2012}
>
```

Dictionary in List

```
users = []

for user_number in range (0,10):
    new_user = {'birth_place': 'Daegu', 'school_name': 'KNU'}
    users.append(new_user)

for user in users[0:5]:
    print(user)
------------------------

{'birth_place': 'Daegu', 'school_name': 'KNU'}
{'birth_place': 'Daegu', 'school_name': 'KNU'}
{'birth_place': 'Daegu', 'school_name': 'KNU'}
{'birth_place': 'Daegu', 'school_name': 'KNU'}
{'birth_place': 'Daegu', 'school_name': 'KNU'}
```

List in Dictionary

```
pizza = {
    'crust': 'thick',
    'toppings': ['mushrooms', 'extra cheese', 'onions'],
    }

print("You ordered a " + pizza['crust'] + "-crust pizza " +
      "with the following toppings:")

for topping in pizza['toppings']:
    print("\t" + topping)
------------------------------------
You ordered a thick-crust pizza with the following toppings:
         mushrooms
         extra cheese
         onions
```

Dictionary in Dictionary

```
users = {'aeinstein': {'first': 'albert',
                       'last': 'einstein',
                       'location': 'princeton'},
         'mcurie': {'first': 'marie',
                    'last': 'curie',
                    'location': 'paris'},
         }

for username, user_info in users.items():
    print("\nUsername: " + username)
    full_name = user_info['first'] + " " + user_info['last']
    location = user_info['location']

    print("\tFull name: " + full_name.title())
    print("\tLocation: " + location.title())
---------------------------------------
Username: aeinstein
           Full name: Albert Einstein
           Location: Princeton

Username: mcurie
Full name: Marie Curie
           Location: Paris
>
```