6주차
파이썬 기초 프로그래밍

튜플, 딕셔너리

튜플

- ( ) 안에 둘러싸인 객체들의 모임
- 시퀀스 자료형이면서 변경 불가능
  - 인덱싱, 슬라이싱, 연결, 반복, 멤버 검사, 길이 정보
- 리스트 보다 접근 속도가 빠름
  - `t = (1, 2, 3)`
  - `t = 1, 2, 3`

```
>>> t = (1,)   # 데이터가 하나인 튜플 정의
>>> t = 1,     # ()가 없어도 쉼표는 꼭 필요
>>> t = tuple(range(10))
```

시퀀스 자료형

- `t = (1,2,3,4,5,6)` 일때

구분	연산	설명	예	결과
인덱싱	[k]	k번 위치의 값	t[0]	1
슬라이싱	[s:t:p]	s부터 t 사이 구간의 값을 p 간격으로 취한다	t[1:5:2]	(2,4)
연결하기	+	두 시퀀스 연결	(1,2)+(3,4,5)	(1,2,3,4,5)
반복하기	*	반복	(1,)*3	(1,1,1)
멤버 검사	in	어떤 값이 시퀀스 자료형에 속하는지 검사	3 in t	True
길이 정보	len()	크기	len(t)	6

튜플 연산

```
>>> t = 1,2,3
>>> t
(1, 2, 3)
>>> t*2
(1, 2, 3, 1, 2, 3)
>>> t+("Happy", "Python")
(1, 2, 3, 'Happy', 'Python')
```

```
>>> print(t[0], t[1:3])
1 (2, 3)
>>> len(t)
3
>>> 2 in t
True
```

튜플 사용 : 배열처럼 사용하기

- 모든 자료 사용하기

```
myTuples = (1,2,3,4,5,6,7,8,9,10)
sum = 0
for t in myTuples :
   sum += t
```

- 일부 자료 사용하기

```
sum = 0
  for i in range(0, len(myTuples), 3) :
    sum += myTuples[i]
```

튜플 사용하기 : 리스트 안의 항목으로 사용

```
from turtle import *

reset()
positions = [(-120, 60, 'blue'), (0, 60, 'black'), (120, 60, 'red'),
             (-60, 0, 'yellow'), (60, 0, 'green')]

for x,y,c in positions :
    up()
    goto(x,y)
    down()
    color(c)
    begin_fill()
    circle(50)
    end_fill()
done()
```

turtle 모듈 import 하기

import turtle

- turtle 모듈을 불러옴.

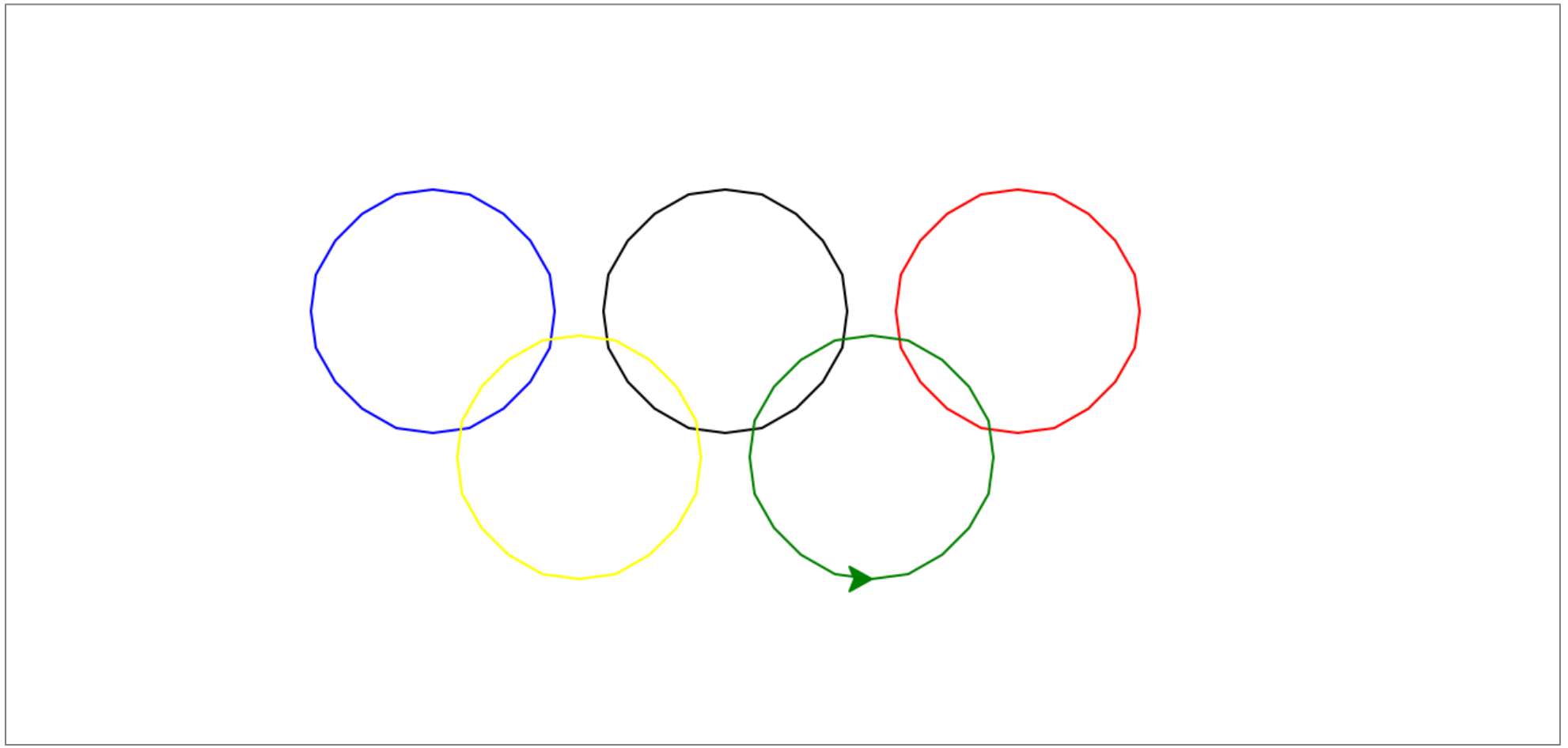
import turtle as t

- turtle 모듈을 불러오고 turtle 대신 t를 사용한다.
- 즉. turtle.forward(100)이 아니라 t.forward(100) 의 형식을 사용한다.

from turtle import *

- turtle 모듈을 불러오고 turtle을 생략한다.
- 즉. turtle.forward(100)이 아니라 forward(100) 의 형식을 사용한다.

실행 결과



튜플 메소드

```
>>> t = (1,2,3,2,3,3)
>>> t.count(2)  # 2가 몇 개 있는가?
2
>>> t.index(2)  # 첫 번째 2의 위치는?
1
>>> t.index(2,1) # 1위치부터 2의 위치 검색
1
>>> t.index(2,2) # 2의 위치부터 2의 위치 검색
3
```

기타

- 튜플의 중첩 : 튜플의 요소로 다른 튜플 사용

```
>>> t = (123, 321, ‘hello’)
>>> u = t, (1,2,3,4,5)
>>> u
((123, 321, ‘hello’), (1,2,3,4,5))
```

- 복수 개 데이터 치환

```
>>> x, y = 1, 2
>>> x, y = y, x
>>> x, y
(2, 1)
```

딕셔너리(dictionary)

- 키, 값의 쌍을 모아 놓은 자료형
- 데이터의 순서가 없이, 키(key)를 이용하여 값(value)을 검색
- 딕셔너리 생성

```
member = {‘basketball’:5, ‘soccer’:11, ‘baseball’:9}
member[‘baseball’] -> 딕셔너리 value 접근: 5

d = {}
d = dict()
d = dict(one=1, two=2)
```

딕셔너리 항목 접근

```
>>> contact = {'Kim':'010123', 'Park':'010234', 'Lee':'010345'}
>>> contact
{'Kim': '010123', 'Park': '010234', 'Lee': '010345'}
>>> number = contact.get('Choi', '010114') # 키기 없으면 두번째 인자 전달
>>> number
'010114'
>>> if 'Park' in contact: # 키가 있는지 확인
...     print('Park이 있습니다.')
...
Park이 있습니다.
```

디셔너리 항목 추가 삭제

- 항목 추가, 수정
  - 딕셔너리[key] = value
    - 딕셔너리에 key:value 항목 추가
    - 예 : contact[“Youn’] = ‘0104567’

key가 존재하면 수정,
없으면 생성

- 항목 삭제
  - 딕서너리.pop(key)
    - 해당 key와 value 항목 삭제
    - 예 : contact.pop(‘Lee’)
  - del 딕셔너리[key]
    - 예 : del contact[‘Youn’]

딕셔너리 메소드

- d = {‘one’:1, ‘two’:2, ‘three’:3}, extra = {‘four’:4, ‘five’:5}

메서드	설명	사용예	결과
keys()	사전의 모든 키 목록 반환	d.keys()	dict_keys(['one', 'three', 'two'])
values()	사전의 모든 값 목록 반환	d.values()	dict_values([1, 3, 2])
items()	(키, 값) 쌍 목록 반환	d.items()	dict_items([('one', 1), ('three', 3), ('two', 2)])
clear()	모든 항목 삭제	d.clear()	{}
copy()	사전 복사(얕은 복사)	d.copy()	{'one': 1, 'two': 2, 'three': 3}
get(key [, x])	값이 존재하면 D[key] 반환. 아니면 x 반환	d.get('one')	1 *(d[‘one’] 과 동일)*
setdefault(key [, x])	값이 존재하지 않을 때 값을 설정	d.setdefault(‘four’,’???’)	‘???’
update()	해당 사전의 모든 항목을 사전에 갱신	d.update(extra)	{'one': 1, 'three': 3, 'two': 2, 'five': 5, 'four': 4}
popitem()	첫번째 (키, 값) 항목을 반환하고 사전에서 제거	d.popitem()	('one', 1) → *{'three': 3, 'two': 2, 'five': 5, 'four': 4}*
pop(key)	key 항목의 값을 반환하고 사전에서 제거	d.pop('two')	2 → *{'three': 3, 'five': 5, 'four': 4}*

for 루프 순회

```
dict_value = {'blue': 10,'yellow': 3, 'red':7}

for key in dict_value :
    print(key)

결과 :
blue
yellow
red
```

```
for (key, value) in dict_value.items() :
    print(key, value)

결과 :
blue 10
yellow 3
red 7
```

리스트를 딕셔너리로 변환하기

```
a=['서울','서울','경기','경기','인천','인천']
dic = dict(zip(range(len(a)), a))
print(dic)

list_A = ['a','b','c']
list_B = [1,2,3]
list_C = [ x for x in zip(list_A,list_B) ]
dic = dict(list_C)
print(dic)

list = [['a', 'b'], ['c', 'd'], ['e', 'f']]
dic = dict(list)
print(dic)
```

```
결과 값 :
{0: '서울', 1: '서울', 2: '경기', 3: '경기', 4: '인천', 5: '인천'}
{'a': 1, 'b': 2, 'c': 3}
{'a': 'b', 'c': 'd', 'e': 'f'}
```

실습 01 : 딕셔너리와 for 문

- 딕셔너리의 모든 항목에 대하여 작업할 때

```
sum = 0
for key in myDictionary :
        sum += myDictionary[key]
        average = sum / len(myDcitionary)
```

```
sum = 0
for (key, value) in myDictionary.items() :
        sum += myDictionary[value]
        average = sum /len(myDcitionary)
```

문제

- 학생 이름과 점수가 저장되어 있는 딕셔너리 pythonClass를 생성하고, class의 평균 점수를 출력하시오.
  - 예 : pythonClass = {“홍길동”:89, “임꺽정”:91, “이순신”:99, “강감찬”:70,……}

실습 02 : 친구 관리

- 친구 관리 프로그램을 수정하여 친구 이름과 전화번호를 함께 입력 받아 저장하시오.

▶ 1. 연락처 목록 출력 -- 친구 이름과 연락처 목록 출력
▶ 2. 연락처 추가 -- 친구 이름과 연락처 추가
▶ 3. 연락처 삭제 -- 이름을 입력하면 연락처까지 삭제
▶ 4. 연락처 변경 -- 이름을 입력하면 수정할 연락처 입력 받음
▶ 5. 종료

실습 03 : 문장 내의 단어 수(word count) 구하기

- 아래와 같은 텍스트 문장에서 단어(word)를 추출하여 빈도수를 출력하는 프로그램을 작성하시오.
  1) 문장 내의 ‘,’와 ‘.’ 문자를 제거한다.
  2) 대문자를 소문자로 바꾼다.
  3) 반복문을 사용하여 단어(key):빈도수(value)를 딕셔너리에 저장한다.
  4) 모든 단어와 빈도수를 출력한다.
- 문장 예 :

```
str = '''GitHub is home to over 50 million developers working
 together to host and review code, manage projects,
 and build software together.'''
```

결과 값 :

```
{'github': 1, 'is': 1, 'home': 1, 'to': 2, 'over': 1, '50': 1, 'million': 1, 'developers': 1, 'working': 1, 'together': 2, 'host': 1,
'and': 2, 'review': 1, 'code': 1, 'manage': 1, 'projects': 1, 'build': 1, 'software': 1}
github 1
is 1
home 1
to 2
over 1
50 1
million 1
developers 1
working 1
together 2
host 1
and 2
review 1
code 1
manage 1
projects 1
build 1
software 1
```

실습 04

- 실습 03의 문장(문자열 str)에서 알파벳의 count를 구하여 출력하시오.
- 결과 값

{'g': 5, 'i': 7, 't': 10, 'h': 5, 'u': 2, 'b': 2, 's': 5, 'o': 13, 'm': 3, 'e': 15, 'v': 3, 'r': 8, '5': 1, '0': 1, 'l': 4, 'n': 5, 'd': 5, 'p': 2, 'w': 3, 'k': 1, 'a': 5, 'c': 2, 'j': 1, 'f': 1}

g 5

i 7

t 10

h 5

u 2

b 2

s 5

o 13

m 3

e 15

v 3

r 8

5 1

0 1

l 4

n 5

d 5

p 2

w 3

k 1

a 5

c 2

j 1

f 1

프로그래밍 과제

- 프로그래밍 과제 04 : 실습 01, 02, 03, 04을 완성하여 클래스 룸에 올리기 바랍니다.
- 제출 마감 : 2020. 10. 18(일) 까지
  - lab01.py
  - lab02.py
  - lab03.py
  - lab04.py