

## LESSON

# 파이썬의 내장 자료형

```
animLength = toTime - fromTime

# Ask user for directory
filePath = c4d.storage.SaveDialog()
filePath, objName = os.path.split(filePath)
objName = objName + ". "
filePath = filePath + "\\ "

# Ask for confirmation
questionDialogText = "Obj Sequence will be saved as:\n\n"\
    "" + filePath + objName + "####.obj\n\n"\
    "From frame " + str(fromTime) + " to " + str(toTime) + " "
proceedBool = c4d.gui.QuestionDialog(questionDialogText)

if proceedBool == True:

    # Loop through animation and export frames
    for x in range(0,animLength):

        # change frame, redraw view
        moveTime = c4d.BaseTime(fromTime,docFps) + c4d.BaseTime(x)
        doc.SetTime(moveTime)
        c4d.EventAdd(c4d.EVENT_FORCEREDRAW)
        c4d.DrawViews(c4d.DRAWFLAGS_FORCEFULLREDRAW)

        # update status bar
        c4d.StatusSetText("Exporting " + str(x) + " of " + str(animLength))
        c4d.StatusSetBar(100.0*x/animLength)

        # get buffer object
        bufferedNumber = str(doc.GetTime().GetFrame(docFps-x))
        if len(bufferedNumber) < 4:
```

## 자료형이란?

- ❖ 프로그래밍 = 자료(Data)를 처리하는 일
- ❖ 파이썬에서 자료를 손쉽게 다룰 수 있도록 하기 위해  
내장 자료형을 제공함

### 숫자(수치) 자료형

- 정수(int)
- 실수(float),
- 복소수(complex)

### 불 자료형

- True
- False

### 군집 자료형

- 문자열(str)
- 리스트(list)
- 튜플(tuple)
- 사전(dict)
- 집합(set)

## 내장 자료형의 특징

### ❖ 자료형의 구분

- 기억 장소의 크기, 저장되는 데이터의 형태, 저장 방식, 값의 범위 등

### ❖ 파이썬은 동적 자료형을 지원

→ 프로그래머가 자료형을 직접 설정할 필요가 없음

- C언어의 경우 같은 숫자라고 해도 int, short, unsigned int, float, double, long 등 메모리나 표현 방식 등에 따라 세분화되지만 파이썬은 그냥 사용하면 됨
- 데이터를 입력하면 데이터 타입을 알아낸 후 그에 맞는 객체를 만들어주면 됨

## 내장 자료형의 분류

분류 기준	종류
데이터 저장 방법	직접 표현, 시퀀스, 매핑
변경 가능성	변경 가능, 불가능
저장 개수	리터럴(한 가지), 컨테이너(여러가지 저장)

## 수치형 자료형 : 정수 -int

- ❖ 소수점이 없는 숫자(양수, 0, 음수)
- ❖ 기본으로 10진수(접두어를 활용해 2, 8, 16진수 등으로 표현 가능)
- ❖ 내장 함수 `int()`를 활용해 정수 자료형으로 변경 가능
- ❖ 범위의 제한이 없음(파이썬 버전 3부터 Long형 또한 정수형으로 통합)

```
a = 0
print(type(a))
print(a)
```

```
<class 'int'>
0
```

```
b = -11
print(type(a))
print(a)
```

```
<class 'int'>
-11
```

```
a = 12345 #10진수
print(type(a))
print(a)
```

```
<class 'int'>
12345
```

```
b = 0b11 # 2진수
print(type(b))
print(b)
```

```
<class 'int'>
3
```

## 수치형 자료형 : 실수 -float

- ❖ 소수점이 있는 숫자
- ❖ 지수 표현 가능(e)
- ❖ 내장 함수 `float()`를 활용해 실수 자료형으로 변경 가능

```
a = float("0.12")  
print(type(a))  
print(a)
```

```
<class 'float'>  
0.12
```

```
b = 2e-4  
print(type(b))  
print(b)
```

```
<class 'float'>  
0.0002
```

```
c = 3e3  
print(type(c))  
print(c)
```

```
<class 'float'>  
3000.0
```

## 수치형 자료형 : 복소수 -complex

- 실수와 허수로 구성된 숫자
- 실수부 + 허수부j

```
a = 10 + 2j  
print(type(a))  
print(a)
```

```
<class 'complex'>  
(10+2j)
```

```
b = 5 - 4j  
print(type(b))  
print(b)
```

```
<class 'complex'>  
(5-4j)
```

## 문자열 자료형

- 문자, 단어 등으로 구성된 문자들의 집합
- 큰 따옴표(“)와 작은 따옴표(‘) 모두 사용 가능
- 내장 함수 str()을 활용해 문자열 자료형으로 변경 가능

```
a = '1'
print(type(a))
print(a)
```

```
<class 'str'>
1
```

```
b = "Hello, World !"
print(type(b))
print(b)
```

```
<class 'str'>
Hello, World !
```

```
c = 12345
c = str(c)
print(type(c))
print(c)
```

```
<class 'str'>
12345
```



## 문자열 자료형

### ❖ 문자열 안에 따옴표를 넣는 방법

- 이스케이프 문자 사용(W)
- 따옴표를 다르게 사용

```
a = "안녕하세요"  
print(a)
```

안녕하세요

```
b = "'안녕하세요'"  
print(b)
```

'안녕하세요'

```
c = '"안녕하세요"'  
print(c)
```

"안녕하세요"

```
d = "\\ '안녕하세요\\'"  
print(d)
```

'안녕하세요'

## 문자열 자료형 : 연산자

### ❖ 연결 연산자(+)와 반복 연산자(\*)

#### 연결 연산자(+)

'1' + '2'



1 + 2

'12'

3

```
a = 'Hello! '  
b = "World"  
print(a+b)
```

Hello! World

#### 반복 연산자(\*)

'2' \* 3



2 \* 3

'222'

6

'안녕하세요' \* 3

'안녕하세요안녕하세요안녕하세요'

## 문자열 자료형 : 인덱싱

### ☛ 선택 연산자

문자	안	녕	하	세	요
인덱스	0	1	2	3	4

```
a = '안녕하세요'  
print(a[0])
```

안

```
a = '안녕하세요'  
print(a[-1])
```

요



문자열은  
시퀀스 자료형으로  
인덱스가 있고,  
인덱스로 값의 접근 가능

## 문자열 자료형 : 슬라이싱

### 범위 선택 연산자

문자	안	녕	하	세	요
인덱스	0	1	2	3	4



변수[시작(이상):끝(미만):스텝]

```
a = '안녕하세요'
print(a[1:3])
```

녕하

```
a = '안녕하세요'
print(a[0:5:2])
```

안하요

## 리스트 자료형

### ❖ 임의의 객체를 순차적으로 저장하는 집합적 자료형

- 문자열이 지닌 대부분의 연산들은 리스트도 지원함
- 대괄호로 정의: `l = [1, 2, 3]`
- 다른 프로그래밍 언어(C, C++) 등과는 달리 동적 배열, 다차원 배열, 인덱싱 등을 훨씬 쉽고 편리하게 사용할 수 있음

```
l = list()
print(l, type(l))

[] <class 'list'>
```

```
: l = [1, 2, 3]
   print(type(l))
   print(l)

<class 'list'>
[1, 2, 3]
```

## 리스트 자료형 : 인덱싱과 슬라이싱

```
l = [1,2,3,4,5,6,7,8,9]
print(l[0])
```

1

```
print(l[0:4])
```

[1, 2, 3, 4]

```
print(l[5])
```

6

```
print(l[len(l)-1])
```

9



인덱스는 0부터 시작하기 때문에  
리스트의 길이를 구한 다음 -1 을 한다.

## 리스트 자료형 : 값의 변경

```
l = [1,2,3,4,5,6,7,8,9]  
l[0] = 99  
print(l)
```

```
[99, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
l[1] = [1,2,3]  
l[2] = "문자"  
print(l)
```

```
[99, [1, 2, 3], '문자', 4, 5, 6, 7, 8, 9]
```



리스트의 요소로  
다른 자료형이 올 수 있다.

## 리스트 자료형 : 함수 활용

```
l.append  
l.clear  
l.copy  
l.count  
l.extend  
l.index  
l.insert  
l.pop  
l.remove  
l.reverse  
l.|
```

```
l = [1,2,3,4,5]  
print(l)  
l.append(6)  
print(l)
```

```
[1, 2, 3, 4, 5]  
[1, 2, 3, 4, 5, 6]
```

```
l = [1,2,3,4,5]  
print(l)  
l.remove(5)  
print(l)
```

```
[1, 2, 3, 4, 5]  
[1, 2, 3, 4]
```



## 튜플 자료형

- ❖ 임의의 객체를 순차적으로 저장하는 집합적 자료형
- ❖ 리스트와 비슷하지만 값을 변경할 수 없는 특징이 있음
- ❖ 소괄호로 정의: `t = (1, 2, 3)`

```
t = tuple()
print(t, type(t))

() <class 'tuple'>
```

```
t = (1, 2, 3)
print(type(t))
print(t)

<class 'tuple'>
(1, 2, 3)
```

## 튜플 자료형 : 특징

❖ 리스트와 비슷한 자료형 : 인덱싱, 슬라이싱 등

```
l = [1,2,3]
t = (1,2,3)
print(l, type(l))
print(t, type(t))
```

```
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
```

```
print(l[0], l[0:2])
print(t[0], t[0:2])
```

```
1 [1, 2]
1 (1, 2)
```

```
print(l + l)
print(t + t)
```

```
[1, 2, 3, 1, 2, 3]
(1, 2, 3, 1, 2, 3)
```

## 튜플 자료형 : 특징

❖ 리스트와의 차이점: 값 변경이 불가능

```
l = [1,2,3]
t = (1,2,3)
print(l, type(l))
print(t, type(t))
```

```
[1, 2, 3] <class 'list'>
(1, 2, 3) <class 'tuple'>
```

```
l[0] = 5
print(l)
```

```
[5, 2, 3]
```

```
t[0] = 5
print(t)
```

```
-----
TypeError                                Tra
<ipython-input-27-0fff53691999> in <module>()
----> 1 t[0] = 5
      2 print(t)
```

```
TypeError: 'tuple' object does not support it
```

## 사전 자료형

- 키를 이용하여 값을 저장하는 자료형
- 정수형 인덱스가 아닌 **키로 값을 저장** → 저장된 자료의 순서는 의미가 없음
- **d = {'a': 1, 'b': 2, 'c': 3}**

```
d = dict()
print(d, type(d))

{} <class 'dict'>
```

```
d = {
    'a' : 1,
    'b' : 2,
    'c' : 3
}
print(type(d))
print(d)
```

```
<class 'dict'>
{'a': 1, 'b': 2, 'c': 3}
```

## 사전 자료형 : 값의 추가/수정

```
d = {'a' : 1, 'b' : 2}  
print(d)
```

```
{'a': 1, 'b': 2}
```

```
d['a'] = 2  
print(d)
```

```
{'a': 2, 'b': 2}
```

```
d['c'] = 3  
print(d)
```

```
{'a': 2, 'b': 2, 'c': 3}
```

```
print(d['d'])
```

```
-----  
KeyError  
<ipython-input-38-c19e1  
----> 1 print(d['d'])
```

```
KeyError: 'd'
```

## LESSON

# 파이썬의 조건문과 반복문

```
animLength = toTime - fromTime

# Ask user for directory
filePath = c4d.storage.SaveDialog()
filePath, objName = os.path.split(filePath)
objName = objName + ". "
filePath = filePath + "\\ "

# Ask for confirmation
questionDialogText = "Obj Sequence will be saved as:\n\n" \
    "" + filePath + objName + "####.obj\n\n" \
    "From frame " + str(fromTime) + " to " + str(toTime) + " "
proceedBool = c4d.gui.QuestionDialog(questionDialogText)

if proceedBool == True:

    # Loop through animation and export frames
    for x in range(0,animLength):

        # change frame, redraw view
        moveTime = c4d.BaseTime(fromTime,docFps) + c4d.BaseTime(x)
        doc.SetTime(moveTime)
        c4d.EventAdd(c4d.EVENT_FORCEREDRAW)
        c4d.DrawViews(c4d.DRAWFLAGS_FORCEFULLREDRAW)

        # update status bar
        c4d.StatusSetText("Exporting " + str(x) + " of " + str(animLength))
        c4d.StatusSetBar(100.0*x/animLength)

        # add buffer 0.001
        bufferedNumber = str(doc.GetTime().GetFrame(docFps))
        if len(bufferedNumber) < 3:
```

## 파이썬의 조건문

- ❖ 프로그램의 실행을 제어하기 위한 제어문 중 하나로 조건에 따라 실행 결과를 달리 할 수 있음
- ❖ 콜론, 들여쓰기 필수!
  - Tab 키를 활용한 들여쓰기 사용 추천

if 조건:

코드

elif 조건:

코드

else:

코드

```
a = 2
if (a == 1):
    print(1)
elif(a == 2):
    print(2)
else:
    print(3)
```

2

```
a = 2
if (a == 1):
    print(1)
else:
    print(2)
```

File "<ipython-input-55-5t  
else:  
^

SyntaxError: invalid syntax

## 파이썬의 조건문

- elif를 활용해 여러 개의 조건 추가 가능
- 조건문 내에 또 다른 조건문을 넣어 중첩 조건 작성 가능
- 조건이 같지 않도록 주의 : 조건은 위에서부터 순서대로 확인하며 내려옴

```
a = 2
if (a == 2):
    print(1)
elif(a == 2):
    print(2)
else:
    print(4)
```

1

조건이 같이 때문에  
두 번째 조건은 무시

```
a = 2
if (a == 1):
    print(1)
elif(a == 2):
    print(2)
elif(a == 3):
    print(3)
else:
    print(4)
```

2

```
a = 1
b = 2
if (a == 1):
    if(b == 3):
        print(1)
    else:
        print(2)
else:
    print(3)
```

2



## for 반복문

- 파이썬의 제어문 중 하나로 **프로그램의 실행을 반복**할 수 있음
- 조건문과 마찬가지로 들여쓰기와 콜론이 매우 중요
- 반복 가능한 객체를 순회하며 반복문 안의 코드를 한번씩 실행



**for** 아이템 **in** 반복 가능한 객체 :

실행 코드

- ① 반복 객체에서 순서대로 하나씩 값을 가져온다.
- ② 아이템에 가져온 값을 담는다.
- ③ 실행 코드를 수행한다.
- ④ 반복 객체가 끝날때까지 순차적으로 반복한다.

## for 반복문

- 리스트, 튜플 등 집합 자료형을 사용하여 요소를 하나씩 가져와 반복 가능
- 반복 가능 객체인 문자열, 사전, 집합 자료형으로도 반복문 사용 가능

```
for i in 10:  
    print(i)
```

-----  
TypeError

```
<ipython-input-27-60bf35fedd47> in <modu  
----> 1 for i in 10:  
      2     print(i)
```

TypeError: 'int' object is not iterable

```
for i in {1,2,3}:  
    print(i)
```

1  
2  
3

```
for i in [1,2,3]:  
    print(i)
```

1  
2  
3

```
for i in "Hello":  
    print(i)
```

H  
e  
l  
l  
o

## while 반복문

- 해당 조건이 참(True)인 경우 반복해서 실행 코드를 반복하는 반복문

while 조건 :  
실행코드



콜론과 들여쓰기에 주의하고  
무한루프에 빠지지 않도록 조건 설정에 주의해야 함

```
In [*]: while(True):  
         print(1)
```

1  
1  
1  
1  
1  
1

```
num = 5  
while (num > 0):  
    print(num)  
    num -= 1
```

5  
4  
3  
2  
1

## while 반복문

- 조건문과 **break** 보조 제어문을 활용해 특별한 조건에 반복문을 완전히 빠져나올 수 있음
- break**를 만나는 순간 제어문을 빠져나가기 때문에 코드 작성에 유의

```
num = 10
while (num > 0):
    if(num == 6):
        print("--end--")
        break
    print(num)
    num -= 1
```

```
10
9
8
7
--end--
```

## while 반복문

- 조건문과 **continue** 보조 제어문을 활용해 특별한 조건에 해당 반복을 건너 뛸 수 있음
- continue**를 만나는 순간 해당 반복을 건너뛰기 때문에 코드 작성에 유의

```
num = 10
while (num > 0):
    print(num, end = ', ')
    num -= 1
    if(num == 6):
        continue
```

10, 9, 8, 7, 6, 5, 4, 3, 2, 1,

```
num = 10
while (num > 0):
    print(num, end = ', ')
    if(num == 6):
        continue
    num -= 1
```

10, 9, 8, 7, 6, 6, 6, 6, 6, 6, 6, 6,  
6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6, 6,  
(무한 반복)