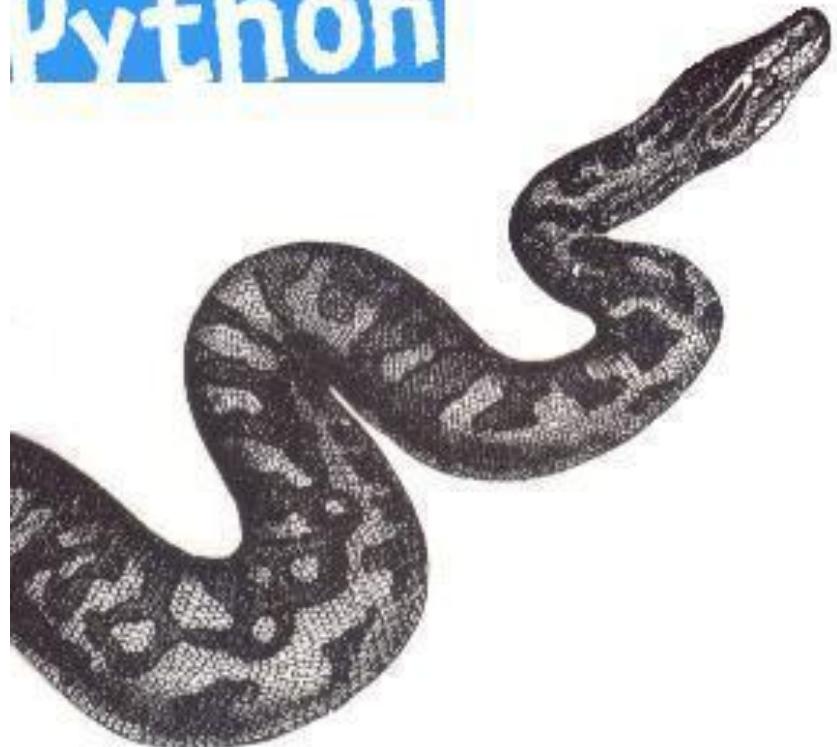


Chapter 01. 강좌 개요

금융데이터수집, 분석에 필요한 파이썬

|파이썬이란?

Python



"A large tropical snake that kills animals for food by winding itself around them and crushing them."

- from Longman Dictionary

|파이썬이란?

- Guido Van Rossum 이 1991년 발표한 프로그래밍 언어
- 코미디 <Monty Python's Flying Circus>



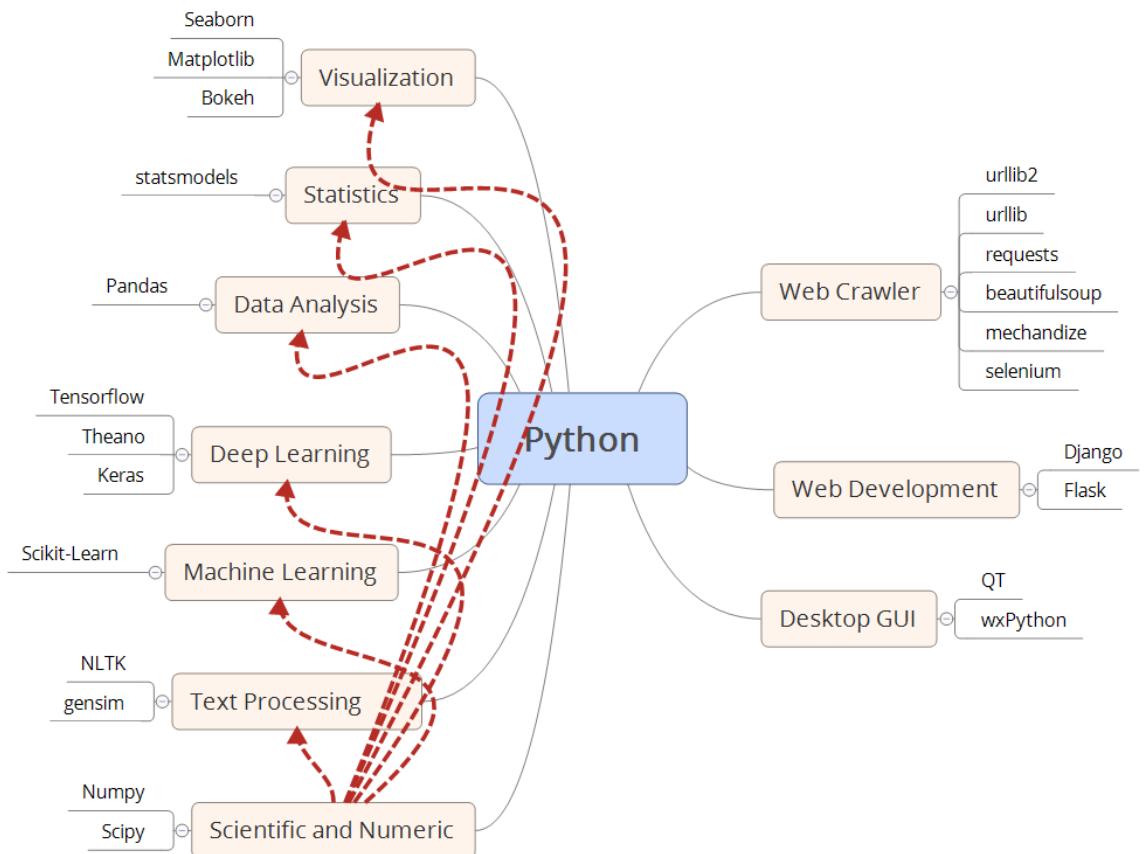
| 파이썬이란?

Python 2와 Python 3 선택하기

- Python은 1991년 처음 탄생한 이후에 많은 발전하였고,
버전 1의 주요 버전인 1.5에서 버전 2로 발전하면서 많은 새로운 개념과 기능이 추가되면서도
하위 호환성을 유지하였습니다.
- 1 버전의 잘못된 문제나 버리고 싶은 문제들도 호환성이라는 이름으로 유지하였지만,
버전 3은 기존 잘못되거나 비효율적인 것들을 정리하고 새롭게 시작한 버전입니다.
- 즉 버전 3부터는 하위 호환성을 유지하지 않습니다.
- Python 2 버전에서 작성된 프로그램을 3버전에서 완벽히 실행할 수 없습니다.
- 2 버전도 2.7을 기준으로 더 이상 새로운 버전이 발표되지 않습니다.
- 보안 및 버그로 인하여 2.7.x 버전으로 업데이트는 진행되나, 기능상 업데이트는 없습니다.
- 현재는 많은 모듈들이 버전 3로 이식되어 있어 본 문서는 Python 3버전으로 Python을 시작 할 것을 권장합니다.

I 파이썬이란?

- 확장성이 뛰어나고, 유지 및 관리하기가 용이
- 언어를 학습하기 쉽고, 가독성이 우수
- 병렬 처리, 람다 함수 지원
- 다양한 기능을 제공하는 라이브러리 제공
 - 수치해석, 기계학습 지원
 - numpy, scipy, scikit-learn, pandas
 - Deep Learning 지원
 - - Tensorflow, Theano, Keras, Pytorch
 - 데이터 시각화
 - matplotlib, seaborn, bokeh
 - Web Framework 지원
 - - Django, Flask
 - Gui 프로그래밍 지원
 - PyQt, wxpython, PyGTK



| 파이썬이란?

- 고속 응용프로그램 개발에 적합하고, 기존의 컴포넌트를 쉽게 연결할 수 있는 접착제 같은 언어로 유용합니다.
- 파이썬의 장점을 다시 간단히 정리하면 아래와 같습니다.
 - 오픈소스
 - 인터프리터 언어
 - 멀티파러다임
 - 다목적
 - 크로스 플랫폼
 - 동적 타입
 - 들여쓰기
 - 가비지 콜렉션

| 파이썬이란?

- 금융에 파이썬이 쓰이는 이유?
 - 파이썬 문법이 금융 알고리즘 개발에 다른 언어보다 쉽다.
 - 파이썬 문법이 수학문법과 비슷하다.
 - numpy 라이브러리를 사용하여 벡터화를 지원한다.
 - 효율성과 생산성이 높다.
 - 다른 언어보다 개발 속도가 빠르다.
 - 시계열 그래프등 시각화가 잘되어 있다.
 - pandas 라이브러리를 활용하면 주식등 쉽게 분석 및 알고리즘을 만들 수 있다.
 - 데이터 수집이 다른 언어에 비해 간단한 코드로 구현이 가능하다.

Chapter 02. 파이썬 프로그래밍 언어
파이썬 프로그래밍 언어 소개

| 이번시간에 배울 내용

1. 파이썬 언어의 소개

- 인터프리터 언어
- PEP 8
- 도움말 보기

2. 특징

- Indent Block
- import
- “Hello Campus” 출력하기
- 주석

I 인터프리터

파이썬은 명령을 실행할 때마다 기계어로 변환하는 인터프리터 언어입니다.

인터프리터(interpreter, 문화어: 해석기)는 프로그래밍 언어의 소스 코드를 바로 실행하는 컴퓨터 프로그램 또는 환경을 말한다. 원시 코드를 기계어로 번역하는 컴파일러와 대비된다. - wikipedia

	컴파일러	인터프리터
기계어 번역 단위	전체	명령 줄 단위
실행 속도	빠름	느림
기계어 번역 속도	느림	빠름

```
#include <stdio.h>
int main()
{
    printf("Hello fastcampus\n");
    return 0;
}
```

<C언어>

```
>>> print ("Hello fastcampus")
Hello fastcampus
<파이썬>
```

I 인터프리터

Life is short, Use Python

아래 그림은 SVM 머신러닝 알고리즘을 Java언어와 파이썬으로 구현한 화면입니다.

```
public class SVMClassifier {
    public static void main(String[] args) {
        SparkConf conf = new SparkConf().setAppName("SVM Classifier Example");
        SparkContext sc = new SparkContext(conf);
        String path = "data/mllib/sample_libsvm_data.txt";
        JavaRDD<LabeledPoint> data = MLUtils.loadLibSVMFile(sc, path).toJavaRDD();

        // Split initial RDD into two... [60% training data, 40% testing data].
        JavaRDD<LabeledPoint> training = data.sample(false, 0.6, 11L);
        training.cache();
        JavaRDD<LabeledPoint> test = data.subtract(training);

        // Run training algorithm to build the model.
        int numIterations = 100;
        final SVMModel model = SVMWithSGD.train(training.rdd(), numIterations);

        // Clear the default threshold.
        model.clearThreshold();

        // Compute raw scores on the test set.
        JavaRDD<Tuple2<Object, Object>> scoreAndLabels = test.map(
            new Function<LabeledPoint, Tuple2<Object, Object>>() {
                public Tuple2<Object, Object> call(LabeledPoint p) {
                    Double score = model.predict(p.features());
                    return new Tuple2<Object, Object>(score, p.label());
                }
            }
        );

        // Get evaluation metrics.
        BinaryClassificationMetrics metrics =
            new BinaryClassificationMetrics(JavaRDD.toRDD(scoreAndLabels));
        double auROC = metrics.areaUnderROC();

        System.out.println("Area under ROC = " + auROC);
    }
}
```

<Java>

```
# Load and parse the data
def parsePoint(line):
    values = [float(x) for x in line.split(' ')]
    return LabeledPoint(values[0], values[1:])

data = sc.textFile("data/mllib/sample_svm_data.txt")
parsedData = data.map(parsePoint)

# Build the model
model = LogisticRegressionWithSGD.train(parsedData)

# Evaluating the model on training data
labelsAndPreds = parsedData.map(lambda p: (p.label, model.predict(p.features())))
trainErr = labelsAndPreds.filter(lambda (v, p): v != p).count() / float(parsedData.count())
print("Training Error = " + str(trainErr))
```

<파이썬>

I PEP 8에 대해서 알아보겠습니다.

파이썬은 PEP(Python Enhancement Proposals) 문서를 제공합니다.

PEP 문서는 아래 주소에서 확인할 수 있습니다.

<https://www.python.org/dev/peps/>

PEP 문서중에서 8번에 해당되는 문서가 파이썬 코드에 대한 스타일 가이드입니다.

- 한 줄의 문자 길이가 79자 이하여야 한다.
- 리스트 인덱스, 함수 호출, 키워드 인수 할당에는 스페이스를 사용하지 않는다.
- 함수의 이름은 소문자로만 한다.
- 변수 할당 앞뒤에 스페이스를 하나만 사용한다.
- 항상 파일의 맨 위에 import 문을 놓는다.
- 한 줄로 된 if문, for와 while 루프, except 문을 쓰지 말고 여러 줄로 나눠서 명확하게 작성한다.

I 도움말 보기

파이썬에서 함수에 대한 도움말을 보고 싶다면 `help()` 함수를 사용하면 됩니다.
`help()` 함수에 알고 싶은 함수의 이름을 전달하면 해당 함수의 설명이 출력이 됩니다.

```
help(print)
```

```
Help on built-in function print in module builtins:
```

```
print(...)  
    print(value, ..., sep=' ', end='\n', file=sys.stdout, flush=False)
```

Prints the values to a stream, or to `sys.stdout` by default.

Optional keyword arguments:

`file`: a file-like object (`stream`); defaults to the current `sys.stdout`.
`sep`: string inserted between values, default a space.
`end`: string appended after the last value, default a newline.
`flush`: whether to forcibly flush the stream.

I 들여쓰기(Indent Block)

파이썬은 들여쓰기에 매우 민감한 언어입니다.

for, while, if와 같은 반복 및 제어문 그리고 사용자 함수, 클래스를 사용하기 위해서는 꼭 필요한 내용입니다.

파이썬은 Indent Block를 기준으로 논리 구조를 파악합니다.

아래 예제는 if 조건문 예제입니다.

```
if a:  
    statement1    #일관된 들여쓰기  
    statement2  
else:  
    statement3  
        statement4 # 일관되지 않은 들여쓰기(에러)
```

1. 가장 바깥쪽에 있는 블록의 코드는 반드시 1열부터 시작.
2. 내부 블록은 같은 거리 만큼 들여 쓰여져야 한다.
3. 블록의 끝은 들여쓰기가 끝나는 부분으로 간주.
4. tab과 space는 섞어서 쓰는 것은 좋지 않다.
5. 들여쓰기 간격은 일정하기만 하면 된다.

Import

파이썬을 처음 실행을 하고 난 뒤에 우리가 할 수 있는 것은?

- 계산기....
- 내장 함수로 인해서 계산기 보다 조금 똑똑한 계산기?

우리가 파이썬을 배우는 목적이 계산기가 아니라면
금융 모형 만들기, 주식 패턴 찾기등의 목적이 있다면
해당되는 기능을 파이썬에게 요청을 해야합니다.

Built-in Functions				
abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

<파이썬 내장 함수>

그 때 사용되는 예약어가 **import** 입니다.

파이썬 표준 라이브러리는 많은 기능있고, 그에 대한 문서는 아래 URL에 설명되어 있습니다.

<https://docs.python.org/ko/3/library/index.html>

우리는 이 과정을 진행하면 표준 라이브러리 뿐만 아니라 pypi에 등록된 라이브러리도 사용해야 합니다.

I "Hello Campus" 출력하기

파이썬에서 화면에 출력하기 위해서는 `print()` 함수를 사용합니다.

여러 값을 쉼표로 구분하여 동시에 출력할 수도 있으며, 각각의 값 사이에는 자동으로 공백 한 개가 추가됩니다.

인수 `sep`를 사용하면 항목 간의 자동으로 생기는 공백 한개를 지정할 수 있습니다.

```
print ("Hello", "fastcampus")
```

Hello fastcampus

```
print ("Hello fastcampus")
```

Hello fastcampus

```
print ("Hello", "fastcampus", sep="_")
```

Hello_fastcampus

I 주석

- 종이로 된 책을 가지고 공부를 할때 본인이 중요하다고 생각되거나 체크하고 싶은 부분에 연필로 적은 기억은 한 번쯤은 있을거라고 생각합니다.
- 시간이 지나서 다시 그 책을 보게 되면 내가 적은 문구의 의미가 다시 떠오르면서 쉽게 내용을 파악할 수 있습니다.

파이썬 언어도 종이책에 표시하듯이 내용을 적을 수 있는 방법을 제공합니다.

이를 주석이라고 부르며, **프로그램 진행에 아무런 영향을 미치지 않습니다.**

파이썬에서 주석은 아래 그림과 같이 # 기호로 시작하면 주석으로 간주합니다.

여러 줄을 주석으로 처리하고 싶다면 쌍따옴표 3개로 시작해서 쌍따옴표 3개로 끝나는 안의 내용은 주석으로 인식합니다.

```
# 아래는 출력을 하는 파이썬 코드입니다.  
print ("Hello fastcampus")
```

```
"""  
이 주석은 여러 줄에 작성한 주석입니다.  
주석은 프로그램에 영향을 미치지 않습니다.  
"""  
print ("Hello fastcampus")
```

Hello fastcampus

| 정리

- 인터프리터
- PEP 문서
- help() 함수
- print() 함수
- 들여쓰기
- 주석

Chapter 03. 파이썬 프로그래밍 언어 변수와 값, 기본 데이터 타입 그리고 출력

| 변수와 값, 데이터 타입, 출력

강의내용

1. 변수, 식별자, 할당문, 예약어
2. 데이터 유형
3. 출력 및 포맷팅

I 변수

- 음식의 종류에 따라서 각기 다른 그릇에 담듯이 데이터도 유형에 따라 담는 그릇이 다릅니다.
 - 변수는 컴퓨터에서 데이터를 담는 그릇입니다.
 - 메모리에 저장된 데이터를 객체라고 하며, 이 객체를 담는 공간을 변수(Variable)이라 합니다.

 - 변수는 **식별자**라고 불리는 이름, 곧 변수명을 정의해야 합니다.
 - 파이썬에서 변수명을 정의할 때 규칙이 존재합니다.
 - 영어 소문자, 대문자, 숫자, _ 으로 구성, 단 **숫자로 시작할 수 없다.**
 - 대소문자는 구별된다.
 - 파이썬이 가지고 있는 키워드는 사용 불가(예약어)
 - 변수명의 예
a, a1, my_name, your_job, MyName, _private, __private_member
- ※ 변수명이 될 수 없는 것들(특수문자는 _만 허용)
- 1abc, @file, fil@e, %x

```

1abc = 3
File "<ipython-input-9-26b7134673f5>", line 1
    1abc = 3
          ^
SyntaxError: invalid syntax

```

I 할당문

- 할당문(assignment statement)는 기호는 `=` 를 사용하여 새 변수를 생성하고 값을 할당합니다.
- 등호 표시가 하나(`=`) 일때는 할당을 의미하고, 두 개(`==`) 일 때는 비교 연산자를 의미합니다.

```
message = "fast campus 퀸트"
pi = 3.14
# new라는 값이 없고, == 를 사용하여 1과 비교하였기 때문에 오류
new == 1
```

```
NameError Traceback (most recent call last)
<ipython-input-13-ecd24a22f45f> in <module>
      1 # new라는 값이 없고, = 를 사용하여 1과 비교하였기 때문에 오류
      2 new == 1
NameError: name 'new' is not defined
```

- 변수에 값을 할당하고 삭제하고 싶다면 `del()` 함수를 사용하면 됩니다.

```
del pi
pi
-----
NameError Traceback (most recent call last)
<ipython-input-30-f84ab820532c> in <module>
----> 1 pi
NameError: name 'pi' is not defined
```

I 예약어

- 파이썬이 가지고 있는 예약어는 변수로 사용할 수 없습니다.
- 예약어란 파이썬에서 미리 정의된 키워드로 그 목록은 아래와 같습니다.
- 파이썬 표준 라이브러리의 keyword 라이브러리에는 예약어를 가지고 있습니다.

```
import keyword

for x in keyword.kwlist:
    print("{}{}".format(x), end=", ")
```

False, None, True, and, as, assert, await, break, class, continue, def, del, elif, else, except, finally, for, from, global, if, import, in, is, lambda, nonlocal, not, or, pass, raise, return, try, while, with, yield,

- 잘못된 변수 예시

```
and = 3

File "<ipython-input-22-788761e0b631>", line 1
    and = 3
          ^
SyntaxError: invalid syntax
```

I 데이터 유형

- 파이썬 자료형
 - 수치형
 - int(정수)
 - float(실수)
 - complex(복소수) : 3+4j, 5+2j
 - 부울형(Bool)
- 데이터 구조
 - str(문자열), list(리스트), tuple(튜플), set(집합), dict(사전)

I 데이터 유형

- 파이썬은 동적 타이핑(Dynamic Typing) 언어로 프로그램 실행시에 변수의 데이터 유형이 결정됩니다.
- `type()` 함수 해당 변수가 어떤 자료형을 가지고 있는지 알려줍니다.
 - 함수의 결과는 <자료형> 형태로 출력

변수명

```
test_variable = 1  
type(test_variable)
```

데이터 유형

int

```
test_variable = 1.0  
type(test_variable)
```

float

```
test_variable = 1j  
type(test_variable)
```

complex

I 데이터 유형

- 같이 해보기
 - $0.1 + 0.2$ 의 결과는?
 - 0.1은 정확하게 이진수로 표현 할 수 없기 때문에 근사값으로 사용합니다.
 - 10진수로 0.1은 2진수로 0.0001100110011001100..... 무한 소수로 표현
 - 그로 인하여 위에서 $0.1 + 0.2$ 값이 우리가 생각했던 0.3이 출력되지 않습니다.
 - 하지만 컴퓨터는 유한 자릿수로 실수를 표현합니다.
 - 곧 표현하지 못하는 자릿수가 존재하여 오차가 생김

I 데이터 유형 변경

- 데이터 유형을 변경하기 위해서는 아래 함수들을 사용하면 된다.
 - 정수형으로 변경 : int()
 - 부울형으로 변경 : bool()
 - 실수형으로 변경 : float()
 - 복수형으로 변경 : complex()

I 출력 및 포맷팅

- 문자열 포매팅 표현식 : 문자열 포맷 코드를 사용하여 출력하는 방법
- 파이썬이 개발된 이후부터 사용된 방식으로 이 형태는 c 언어의 printf 함수를 기반으로 합니다.
- 지금도 여전히 기존에 개발된 코드에서 쉽게 찾아 볼 수 있는 형태입니다.

```

1 year = 2018
2 month = 6
3 day = 2
4 today = "토요일"
5 print ("%d년 %d월 %d일 %s입니다." % (year, month, day, today))

```

2018년 6월 2일 토요일입니다.

```

year = 2018
month = 6
day = 2
today = "토요일"
print ("%f년 %f월 %f일 %s입니다." % (year, month, day, today))

```

2018.000000년 6.000000월 2.000000일 토요일입니다.

```

year = 2018
month = 6
day = 2
today = "토요일"
print ("%x년 %x월 %x일 %s입니다." % (year, month, day, today))

```

7e2년 6월 2일 토요일입니다.

코드	설명
%s	문자열 (String)
%c	문자 1개(character)
%d	정수 (Integer)
%f	부동소수 (floating-point)
%o	8진수
%x	16진수
%%	Literal % (문자 % 자체)

print (" 코드 blah, blah, blah") % (변수)

I 출력 및 포맷팅

- 문자열 포매팅 메서드 호출 : "{}...{}".format()
- 파이썬 2.6과 3.0에 새롭게 추가된 방법입니다.
- 이 형태는 C#/.NET에서 제공되는 같은 이름을 가진 도구에서 일부 파생되었다고 합니다.
- 아래 예제를 통하여 사용법을 알아봅시다.
 - "{}.....{}"에서 ""으로 감싸여 있는 부분을 문자열이라고 합니다.
 - 이 문자열에는 format이라는 출력을 담당하는 기능을 가졌습니다.
 - "{}.... {}".format(첫번째 위치, 두번째 위치)

```
year = 2018
month = 6
day = 2
today = "토요일"
print ("{}년 {}월 {}일 {}입니다.".format(year, month, day, today))
```

2018년 6월 2일 토요일입니다.

year	month	day	today
------	-------	-----	-------

I 정리

- 변수명 정의
- 할당문
- 예약어 리스트
- 동적 타이핑
- 파이썬의 데이터 유형 목록
- 출력 및 포맷팅 방식 2가지

Chapter 04. 파이썬 프로그래밍 언어 연산자

|파이썬이란?

강의내용

1. 기본 숫자 연산자
2. 비교 연산자
3. 논리 연산자
4. 연산자 우선순위

I 연산자

파이썬에서 사용하는 연산 기호는 아래와 같습니다.

```
# 더하기
print(10000 + 1234)

# 빼기
print(10000 - 1234)

# 곱하기
print(10000 * 23)

# 나누기
print(10000 / 8)

# 제곱
print(2 ** 3)

# 나머지
print(50 % 8)
```

Operator	Description
%	나머지를 반환
**	지수연산을 수행
//	나눗셈의 소수점 이하는 제거

11234
8766
230000
1250
8
6
2

| 연산자

256 / 8 의 결과 값은 어떤 데이터 유형일까요?

- 정수형을 정수형으로 나눗셈을 하면 위의 예제처럼 32라는 정수형이 나옵니다.
- $256 / 7$ 은 36.571 이라는 실수형이 나옵니다.
- 두 가지 경우만 봐도 정수형을 정수형으로 나눗셈을 할 때 딱 나눠지는 경우와 그렇지 않은 경우가 발생합니다. 그래서 파이썬에서는 3.0 이상 버전에서는 정수와 정수의 나눗셈의 결과는 실수형으로 결과가 나옵니다.

I 비교 연산자

비교 연산자의 종류는 아래와 같습니다.

연산자	설명	예시	결과
<code>==</code>	같다면 True	<code>1==2</code>	
<code>></code>	크면 True	<code>1>2</code>	
<code>>=</code>	크거나 같다면 True	<code>1>=2</code>	
<code><</code>	작으면 True	<code>1<2</code>	
<code><=</code>	작거나 같으면 True	<code>1<=2</code>	
<code>!=</code>	다르면 True	<code>1 != 2</code>	

비교 연산자의 결과값은 부울(Boolean)형으로 반환됩니다.

연속적인 비교도 할 수 있습니다.

`1 < 2 < 3 < 4 < 5 < 6 < 7`

`5 == 5 < 10`

I 논리 연산자

논리 연산자의 종류는 아래와 같습니다.

연산자	설명	연산자 우선순위
and	두 구문 모두 참일 경우에는 True 아니면 False	2
or	두 구문 중 하나라도 참인 경우 True 둘다 거짓이면 False	3
not	구문이 참이면 거짓, 거짓이면 참으로 값을 뒤집는다.	1

- Boolean 연산자는 모든 프로그래밍에서 매우 자주 사용됩니다.
- Boolean 연산자는 **and**와 **or, not** 이 있습니다. 이 연산자는 두 구문을 비교하거나 Boolean 값을 뒤집습니다.
- 비교 연산자와 마찬가지로 결과 값은 항상 Boolean 값이 됩니다.

True and True는 True
True and False는 False
False and True는 False
False and False는 False
True or True는 True
True or False는 True
False or True는 True
False or False는 False
Not True는 False
Not False는 True

I 논리 연산자

아래 문제를 머리속으로 풀어봅시다.

not False

True and False

False or True

True and False or not False

I 연산자 우선순위

지금까지 이야기한 연산자는 서열을 가지고 있습니다.

파이썬 연산자 우선순위를 정리하면 아래와 같습니다.

높은 우선순위에서 낮은 순위로.....

연산자	설명
**	지수승
~	비트반전
+x, -x	양수, 음수
*, /, %	곱셈, 나눗셈, 나머지
+, -	덧셈, 뺄셈
<<, >>	쉬프트
&	비트 AND
^	비트 XOR
	비트 OR
in, not in, is, is not, <, <=, >, >=, <>, !=, ==	비교연산자, 시퀀스 요소 검사
not x	Boolean NOT
and	Boolean AND
or	Boolean OR

| 정리

- 기본 연산자
- 비교 연산자
- 논리 연산자
- 연산자 우선순위

Chapter 05. 파이썬 프로그래밍 언어 날짜와 시간

| 이번 시간에 배울 내용

1. 날짜와 시간에 관한 모듈
2. datetime
3. dateutil

I 표준 모듈(Standard Library)

- 파이썬에서 파일을 찾거나, 시간을 확인하거나, 난수를 생성하거나 그런 일들을 하는 모듈이 존재하는데 이를 표준 라이브러리라고 합니다.(모듈 설명에서 한번 언급했습니다.)
- 이번 시간에는 표준 라이브러리에서 시간과 날짜 모듈에 대해서 알아보겠습니다.
 - 시간과 날짜 표준 라이브러리 모듈 이름 : `datetime`

객체이름	용도
<code>date</code>	날짜 다루기
<code>time</code>	시각 다루기
<code>datetime</code>	일시 다루기
<code>timedelta</code>	두 일시의 차 다루기

The screenshot shows a section of the Python 3.6.1 documentation titled "The Python Standard Library". The main content area is titled "The Python Standard Library" and discusses the standard library's extensive range of components. Below the main content, there is a sidebar with links to "Previous topic" (10. Full Grammar specification), "Next topic" (1. Introduction), and "This Page" (Report a Bug, Show Source). At the top of the page, there is a navigation bar with "Python » 3.6.1 » Documentation »".

- `datetime` 모듈을 사용하면 문자열로 된 날짜를 `datetime` 객체로 변환하고, 반대로 변환할 수 있습니다.
- 주차 계산, 요일 계산 등을 쉽게 할 수 있어서 시계열 분석에 활용할 수 있습니다.

I datetime 모듈

- date 객체는 날짜(연, 월, 일)를 취급합니다.

메서드이름	설명	반환값
date(year, month, day)	지정한 날짜의 date 객체를 생성	datetime.date
today()	오늘 날짜의 date 객체를 생성	datetime.date
weekday()	월요일을 0, 일요일을 6으로 하여 요일을 반환	int
isoweekday()	월요일을 1, 일요일을 7로 하여 요일을 반환	int
isoformat()	ISO 8601 형식(YYYY-MM-DD)의 날짜 문자열 반환	str
strftime(format)	지정한 포맷에 따라 날짜 문자열을 반환	str
__str__()	isoformat()과 같은 결과를 반환	str

객체이름	설명	반환값
year	년 값을 반환	int
month	월 값을 반환	int
day	일 값을 반환	int

```
# date 샘플 코드
from datetime import date
```

```
newyearsday = date(2017, 6, 3)
```

```
newyearsday
```

```
datetime.date(2017, 6, 3)
```

```
newyearsday.year, newyearsday.month, newyearsday.day
```

```
(2017, 6, 3)
```

```
newyearsday.weekday()
```

```
5
```

```
newyearsday.isoformat()
```

```
'2017-06-03'
```

```
str(newyearsday)
```

```
'2017-06-03'
```

```
newyearsday.strftime('%Y/%m/%d')
```

```
'2017/06/03'
```

```
newyearsday.strftime('%Y/%m/%d (%a)')
```

```
'2017/06/03 (Sat)'
```

```
date.today()
```

```
datetime.date(2017, 6, 2)
```

I datetime 모듈

- datetime 객체는 시각을 다룬다. 시분초뿐만 아니라 마이크로초도 포함된다.

메서드이름	설명	반환값
time(hour=0, minute=0, second=0, microsecond=0, tzinfo=None)	지정한 시각의 time 객체를 생성	datetime.time
today()	기본 표준시간대의 현재 일시를 반환	datetime.datetime
date()	같은 연월일의 date 객체를 반환	datetime.date
time()	같은 시분초의 time 객체를 반환	datetime.time
isoformat()	ISO 8601 형식	str
strftime(format) strptime(format)	지정한 포맷에 따라 날짜 문자열을 반환 지정한 포맷에 따라 날짜 객체로 반환	str datetime.datetime
__str__()	isoformat()과 같은 결과를 반환	str
tzname()	표준시간대 이름의 문자열을 반환	str

객체이름	설명	반환값
year	년 값을 반환	int
month	월 값을 반환	int
day	일 값을 반환	int
hour	시 값을 반환	int
minute	분 값을 반환	int
second	초 값을 반환	int
microsecond	마이크로초 값을 반환	int
tzinfo	표준시간대 정보	객체

```
from datetime import datetime
```

```
today = datetime.today()
```

```
today.date()
```

```
datetime.date(2017, 6, 2)
```

```
today.time()
```

```
datetime.time(14, 8, 18, 88234)
```

```
today.isoformat()
```

```
'2017-06-02T14:08:18.088234'
```

```
today.strftime("%Y-%m-%d %H:%M:%S")
```

```
'2017-06-02 14:08:18'
```

I 날짜 변경 포맷 형식

- strftime() 메소드를 사용하면 datetime 객체를 문자열로 변경할 수 있다고 설명했습니다.
- 변경시에 사용될 format을 정리했습니다.

%Y	Year with century as a decimal number.	0001, 0002, ..., 2013, 2014, ..., 9998, 9999
%m	Month as a zero-padded decimal number.	01, 02, ..., 12
%d	Day of the month as a zero-padded decimal number.	01, 02, ..., 31
%H	Hour (24-hour clock) as a zero-padded decimal number.	00, 01, ..., 23
%M	Minute as a zero-padded decimal number.	00, 01, ..., 59
%S	Second as a zero-padded decimal number.	00, 01, ..., 59

I dateutil 모듈

- dateutil 모듈은 datetime 모듈에 대한 확장 기능을 제공합니다.
 - 다양한 문자열 형식의 날짜 구문
 - 날짜의 차이 계산
- 별도로 설치를 하여야 사용할 수 있습니다.
 - pip install python-datetime
- relativedelta 모듈을 사용하면 날짜 사이의 차이를 계산할 수 있다.
 - relativedelta(datetime1, datetime2)

I dateutil 모듈

```
from dateutil.relativedelta import relativedelta
from datetime import datetime, date

now = datetime.now()

now + relativedelta(months =+ 1)
datetime.datetime(2017, 7, 2, 15, 24, 47, 659553)

now + relativedelta(months =- 1, weeks =+1)
datetime.datetime(2017, 5, 9, 15, 24, 47, 659553)

from dateutil.relativedelta import MO, TU, WE, TH, FR, SA, SU

now + relativedelta(weekday = SA)
datetime.datetime(2017, 6, 3, 15, 24, 47, 659553)

now + relativedelta(day = 30, weekday = SA(-1))
datetime.datetime(2017, 6, 24, 15, 24, 47, 659553)

date(2017,1,1) + relativedelta(yearday = 100)
datetime.date(2017, 4, 10)

relativedelta(now, date(2017,10,10))

relativedelta(months=-4, days=-7, hours=-8, minutes=-35, seconds=-13, microseconds=+659553)
```

time 모듈

- time 모듈의 sleep 메소드를 사용하면 프로그램의 동작을 정해진 시간 만큼 정지할 수 있습니다.

```
from time import sleep
```

```
for x in range(0,10):  
    print(x)  
    sleep(1)
```

```
0  
1  
2  
3  
4  
5  
6  
7  
8  
9
```

- 데이터 수집시 프로그램의 동작을 잠시 정지하고 싶을 때 사용합니다.

| 정리

- datetime의 사용법
- 문자열 <-> datetime 객체
- 시간 차이 계산
- sleep

Chapter 06. 파이썬 프로그래밍 언어

문자열

I 문자열

강의내용

1. 문자열
2. 이스케이프문자
3. 논리 연산자
4. 연산자 우선순위

I 문자열이란?

문자열(string)이란 큰따옴표 혹은 작은따옴표안에 표현된 자료형(str)입니다.

```
text1 = "안녕하세요."
text2 = '안녕하세요'
text3 = """안녕하세요.
수강생 여러분 반갑습니다.
파이팅!!"""

```

- text1은 큰따옴표로 만들어진 문자열입니다.
- text2는 작은 따옴표로 만들어진 문자열입니다.
- text3은 여러 줄을 입력한 문자열입니다. 여러 줄을 입력하고 싶다면 큰따옴표 3개로 시작해서 3개로 끝나면 됩니다.
 - 혹은 작은 따옴표 3개로 시작해서 작은 따옴표 3개로 끝나면 됩니다.
- 한가지 주의사항은 큰따옴표로 시작했다면 큰 따옴표, 작은 따옴표로 시작했다면 작은 따옴표로 끝나야합니다.

```
text4 = '작은 따옴표와 큰 따옴표를 혼합해서 사용하면...'
File "<ipython-input-4-2c048bcaca4d>", line 1
    text4 = '작은 따옴표와 큰 따옴표를 혼합해서 사용하면...'^
SyntaxError: EOL while scanning string literal
```

| 이스케이프문자란?

- 이스케이프 문자는 [이스케이프 시퀀스](#)를 따르는 문자들로서 다음 문자가 특수 문자임을 알리는 백슬래시(\)를 사용한다.
일부 제어 시퀀스인 이스케이프 문자들은 미리 예약 되어있다. – Wikipedia
- 문자열 안에 작은 따옴표나 큰 따옴표를 쓰고 싶다면 이스케이프 문자를 사용하면 됩니다.

이스케이프 문자	의미	예제
\'	문자열 안에 작은 따옴표 표현	<pre>print ('Chanwoong\' fastcampu')</pre> Chanwoong' fastcampu
\"	문자열 안에 큰 따옴표 표현	<pre>print ('Chanwoong say \"Hi\"')</pre> Chanwoong say "Hi"
\n	줄바꿈	<pre>print ('Chanwoong \nseo')</pre> Chanwoong seo
\t	문자 사이의 tab 간택	<pre>print ('Chanwoong \tseo')</pre> Chanwoong seo

윈도우 언어 설정에 따라서 역슬래시 표현이 원화(₩)로 출력
표현은 원화 표시라도 실제 파일에서 역슬래시로 동작

| 시퀀스 자료형이란?

- 여러 객체를 저장하고 있습니다.
- 각 객체들은 순서를 가지고 있습니다.
- 각 객체들은 첨자를 이용하여 접근이 가능합니다.

✓ 종류

- 리스트, 튜플, 문자열

✓ 공통연산

- 인덱싱(indexing)
- 슬라이싱(slicing)
- 연결하기
- 반복하기
- 멤버십 테스트
- 길이정보

I 인덱싱(indexing)

- 인덱싱은 인덱스 번호를 사용하여 특정 위치에 있는 데이터에 접근하는 것을 의미합니다.
- 문자열 안의 각 문자는 위치가 정해져 있습니다. 이를 인덱스 번호로 접근할 수 있습니다.
- 문자열 색인은 0부터 시작하며, 첫 번째 문자의 색인은 0이고 두번째 문자는 1입니다.

문자열	f	a	s	t			c	a	m	p	u	s
양수 Index	0	1	2	3	4	5	6	7	8	9	10	
음수 Index	-11	-10	-9	-8	-7	-6	-5	-4	-3	-2	-1	

- ✓ len() 함수는 시퀀스형 자료안의 전체 요소의 개수를 알려주는 함수입니다.

```
len("fast campus")
```

I 슬라이싱(Slicing)

- 시퀀스 자료형의 일정 영역에서 새로운 객체를 반환하며, 결과의 자료형은 원래의 자료형과 동일합니다.
- 슬라이싱은 세가지 값을 가진다.

[start : stop : step]

- start 번호 ~ stop 전 번호까지 선택된다. 예를 들면 [1:5]라고 하면 인덱스 1,2,3,4까지 선택된다.

- 세 번째 step은 데이터를 취하는 간격이고 이를 확장 슬라이싱이라 합니다.
- 세상에서 가장 긴 영어 단어 : Pneumonoultramicroscopicsilicovolcanoconiosis
- 위의 단어를 문자열로 표현하면 아래와 같은 인덱스를 가집니다.

P	n	e	u	o	s	i	s
0	1	2	3	41	42	43	44
-45	-44	-43	-42	-4	-3	-2	-1

| 슬라이싱(Slicing) 연습

```
tmp = "Pneumonoultramicroscopicsilicovolcanoconiosis"
```

```
tmp[0:5]
```

```
'Pneum'
```

```
tmp[10:20]
```

```
'tramicrosc'
```

```
tmp[10: ]
```

```
'tramicroscopicsilicovolcanoconiosis'
```

```
tmp[ :-1]
```

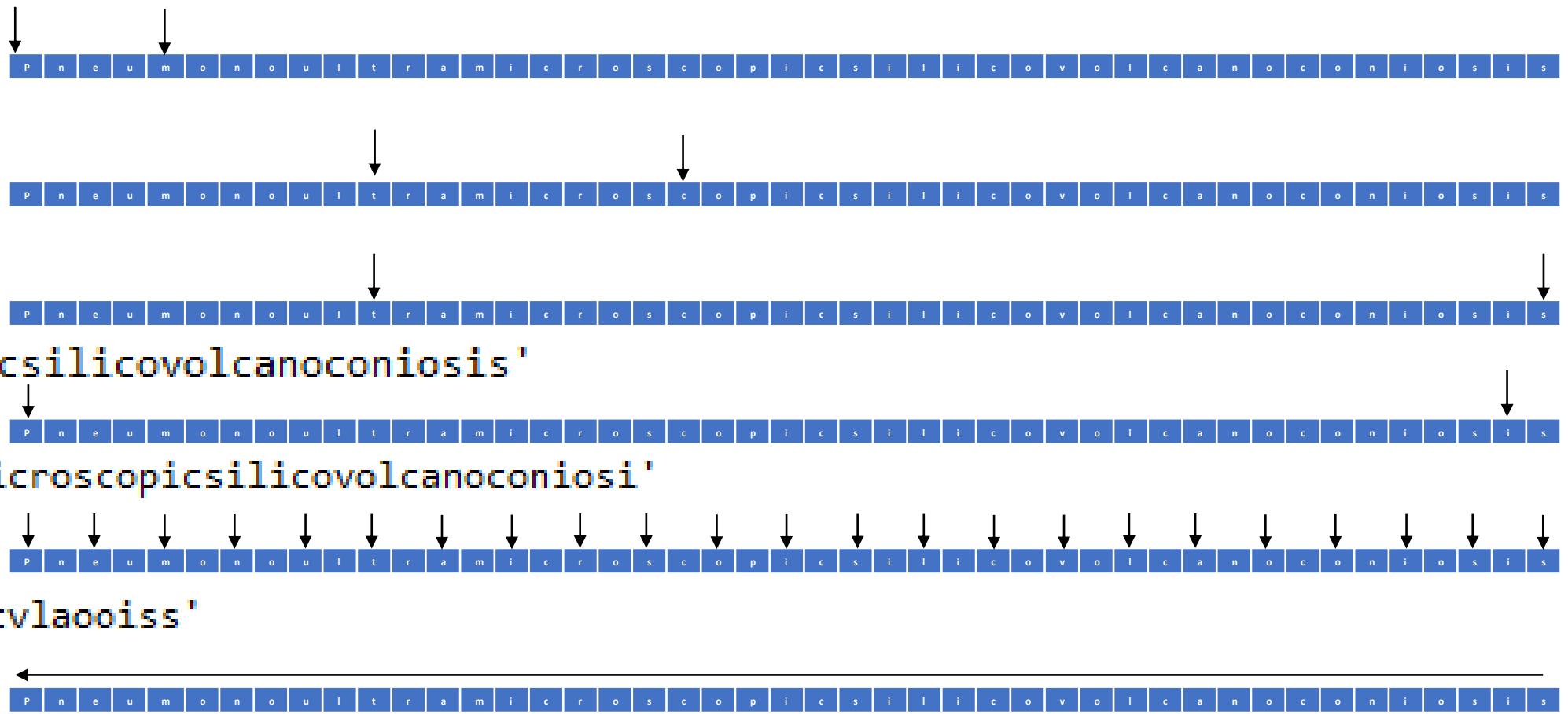
```
'Pneumonoultramicroscopicsilicovolcanoconiosi'
```

```
tmp[::2]
```

```
'Pemnutairsoislsclvlaooiss'
```

```
tmp[:: -1]
```

```
'sisoinoconaclovociliscipocsorcimartluuonomuenP'
```



I 연결 및 반복하기

- 시퀀스형 자료형의 특징은 각각의 시퀀스 변수들을 + 연산자를 사용하여 연결할 수 있습니다.
- * 연산자를 사용하여 데이터를 반복해서 출력할 수 있습니다.

```
"Hello" + "fastcampus"
```

```
'Hellofastcampus'
```

```
"Hello" * 6
```

```
'HelloHelloHelloHelloHelloHello'
```

- 위의 예제는 + 연산자를 사용해서 두 문자열을 하나로 합치고, * 연산자를 사용하여 문자열을 반복한 예제입니다.
- 아래는 모모랜드의 뽐뽀 가사의 일부입니다. 아래 코드가 이해가 된다면....

Give it to you, my 눈눈눈눈눈 눈빛
 쏟아지는 my t-t-t-t-t-touch
 하나뿐인 my lo-lo-lo-lo-love, my lover

```
"Give it to you, my " + "눈" * 6 + " 눈빛"
```

```
'Give it to you, my 눈눈눈눈눈 눈빛'
```

```
"쏟아지는 my " + "t-" * 6 + "touch"
```

```
'쏟아지는 my t-t-t-t-t-touch'
```

```
"하나뿐인 my " + ("lo-" * 5)[:-1] + "ve" + ", my lover"
```

```
'하나뿐인 my lo-lo-lo-lo-love, my lover'
```

| 멤버십 테스트

- ✓ in 연산자를 사용하여 대상이 되는 변수 안에 사용자가 질의하는 텍스트가 포함되어 있는지 확인할 수 있습니다..

```
text = "fastcampus"  
'cam' in text
```

True

```
'test' in text
```

False

```
'test' not in text
```

True

I 문자열 메소드

- 문자열은 정확히 말하면 문자열 클래스입니다.
- 클래스안에 구현된 함수들을 메소드라고 부릅니다.
- 문자열 메소드는 텍스트 데이터를 전처리할 때 유용하게 사용할 수 있습니다. 아래는 자주 사용하는 메소드만 적었습니다.

- 특정 문자열 개수 세기 – count()
- 대소문자 변환 - upper(), lower() – 대소문자 변환, capitalize() – 첫 문자를 대문자로
- 검색 관련 - count(s) – 문자열 s가 몇 번 발생한지 카운트, find(s), rfind(s) – s위치, 없으면 -1
index(s) – s위치, 없으면 예외 ValueError 발생
- 편집 및 치환
 - strip(), lstrip(), rstrip() – 좌우 공백 없앰
 - replace(a, b) – a를 b로 바꾼다
 - expandtabs() – 탭을 공백 문자로 바꾼다
- 분리와 결합
 - split() – 문자열 분리
 - join() – 문자열 결합

| 정리

- 문자열 정의
- 이스케이프 문자
- 시퀀스형 자료형
- 인덱스, 인덱싱, 슬라이싱
- 문자열의 연결과 반복
- 멤버십 테스트
- 문자열 메소드

Chapter 07. 파이썬 프로그래밍 언어 자료구조

| 이번시간에 공부할 내용

1. 리스트(list)
2. 튜플(tuple)
3. 사전(dictionary)
4. 리스트, 사전 정렬
5. 집합(set)

I 리스트

- list는 원소를 주어진 순서대로 보관하는 데이터 유형입니다.
- list에 보관하는 원소나 값은 매우 다양한 데이터 타입이 될 수 있습니다.
- list 안에 list를 보관할 수도 있습니다.
- list는 열차처럼 열차 한량 한량들이 모여서 전체 열차를 구성하듯 데이터가 모여 list 전체를 구성 합니다.
- 열차에 사람이나, 화물, 자동차, 탱크를 수송할 수 있듯이 list 또한 파이썬의 데이터 유형을 저장할 수 있습니다.
- **리스트는 시퀀스형 자료형입니다. 시퀀스형 자료형의 특성을 가지고 있습니다.**



I 리스트

- list를 생성해 보겠습니다.
 - list 이름 = [원소1, 원소2, 원소3..... 원소N]
 - 빈 list 만들 때는 아래와 같습니다.

- list 이름 = list()
- list 이름 = []

```
stock = ['삼성전자', '엘지전자', 'SK하이닉스']
```

- 인덱스를 사용하여 값을 출력할수 있습니다.
- 원하는 인덱스의 값을 변경하고 싶을 때는 시퀀스 접근 방식으로 접근하여 할당 연산자(=)를 사용하여 대입합니다.

```
print (stock[1])
```

엘지전자

```
stock[1] = '현대자동차'
```

```
print (stock[1])
```

현대자동차

I 리스트

- list는 스퀀스형 자료형입니다.
- 앞의 예제의 데이터를 그림으로 표현하면 아래와 같습니다.

stock	삼성전자	현대자동차	SK하이닉스
양의 index	0	1	2
음의 index	-3	-2	-1

- for문을 사용해서 출력을 해보겠습니다.

```
for x in stock:
    print(x)
```

삼성전자
현대자동차
SK하이닉스

```
for x in stock[::-1]:
    print(x)
```

SK하이닉스
현대자동차
삼성전자

I 리스트

- list에 새로운 값을 추가해보겠습니다.
 - list에 하나의 데이터를 추가하고 싶다면 **append()** 메소드를 사용하여 데이터를 입력합니다.

```
stock.append('gs')
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스', 'gs']
```

```
del stock[3]
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스']
```

del() 함수를 사용하면

- 원하는 인덱스의 데이터를 삭제할 수 있습니다.
- 전체 list를 삭제할 수도 있습니다.

- list의 마지막 뒤에 여러 개의 데이터를 입력하고 싶다면 **extend()** 메소드를 사용합니다.
- extend() 메소드의 인자값은 리스트 유형으로 전달합니다.

```
stock.extend(['롯데제과', '신세계'])
```

```
stock
```

```
['삼성전자', '현대자동차', 'SK하이닉스', '롯데제과', '신세계']
```

- insert() 메소드를 사용하면 원하는 인덱스에 데이터를 저장할 수 있습니다.

```
stock.insert(2, '엘지전자')
```

```
stock
```

```
['삼성전자', '현대자동차', '엘지전자', 'SK하이닉스', '롯데제과', '신세계']
```

I 리스트

- slicing으로 데이터를 접근해보겠습니다.

```
stock[1:3]
```

```
['현대자동차', '엘지전자']
```

```
stock[:2]
```

```
['삼성전자', '현대자동차']
```

```
stock[::-2]
```

```
['삼성전자', '엘지전자', '롯데제과']
```

- 문자열에서 학습한 시퀀스 특성 그대로 list에서도 적용됩니다.
- index() 메소드를 사용하면 list의 데이터 중 같은 값이 있다면 처음 발견한 위치의 index 값을 반환합니다.
- 하지만 값이 없다면 ValueError를 일으킵니다.

```
stock.index('엘지전자')
```

```
2
```

```
stock.index("한화")
```

ValueError

```
<ipython-input-35-8013a3b60418> in <module>
----> 1 stock.index("한화")
```

Traceback (most recent call last)

ValueError: '한화' is not in list

I 리스트

- 중첩 list에 대해서 알아보겠습니다.
- list안에 데이터는 사용자가 원하는 값을 저장할 수 있다고 이야기 했습니다.
- 당연히 list안에 list를 저장할 수 있습니다.

```
stock2 = [['삼성전자', '삼성화재', '삼성생명'], ['엘지전자', '엘지화학'], ['SK하이닉스', 'SK에너지']]
```

- 중첩 list에서 데이터의 접근도 [][]을 이용하면 됩니다.

stock2[0][1]

'삼성화재'



I 리스트

- 자주 사용하는 메소드

메소드	설명
append	자료를 리스트 끝에 추가 (혹은 스택의 push)
insert	자료를 지정된 위치에 삽입
index	요소 검색(Search)
count	요소 개수 알아내기
sort	리스트 정렬
reverse	자료 순서 바꾸기
remove	지정 자료 값 한 개 삭제
pop	리스트의 마지막 값을 읽어내고 삭제 (스택의 pop)
extend	리스트를 추가

I 리스트

- list의 데이터 정렬하기
 - sort() 메소드를 사용하면 데이터를 정렬할 수 있습니다.
 - 역순으로 정렬할땐 reverse 옵션을 사용합니다.
- sort() 메소드를 사용할 때 참고할 점
 - sort 메소드는 key 옵션을 사용하여 문자열을 int형으로 인식하게 만들고 정렬합니다.
 - reverse option를 사용하면 역순으로 정렬합니다.

```
L = ['123', '34', '56', '23456']
```

```
L.sort()
```

```
L
```

```
['123', '23456', '34', '56']
```

```
L.sort(key=int)
```

```
L
```

```
['34', '56', '123', '23456']
```

```
L.sort(reverse=True)
```

```
L
```

```
['56', '34', '23456', '123']
```

```
L.sort(reverse=True, key=int)
```

```
L
```

```
['23456', '123', '56', '34']
```

I 리스트

- list comprehension(리스트 내장) 이란?
 - list를 쉽게 생성하기 위한 기능입니다.
 - 0부터 9까지 숫자를 제곱승을 구해서 list에 저장하는 코드를 살펴보겠습니다.
 - list comprehension 문법은 아래와 같습니다.
- [expression for expr in sequence1
 for expr2 in sequence2 ...
 for expN in sequenceN
 if condition]
- 위의 문법을 사용해서 compre_1과 동일한 데이터를 가진 list를 만들어보겠습니다.

```
compre_1 = list()
for x in range(0,10):
    compre_1.append(x**2)
```

```
compre_1
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

표현식	for expr in seqence
compre_2 = [x**2]	for x in range(0,10)]

```
compre_2
```

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

I 리스트

- list comprehension를 사용하여 데이터를 생성할 때 조건문을 넣을 수 있습니다.
- 아래 예제는 0 ~ 29까지 숫자 중에서 짝수만 tmp라는 list에 저장합니다.

```
tmp = [k for k in range(0,30) if k % 2 == 0]
```

```
tmp
```

```
[0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20, 22, 24, 26, 28]
```

- 아래 예제는 구구단에서 짝수단만 저장하는 예제입니다.

```
[x * y for x in range(2,10) for y in range(1,10) if x % 2 == 0]
```

I 리스트 문제

- 문제 1 : 앞에서 만든 윤년을 list comprehension 방식으로 저장해 봅시다.

- 문제 2 : 문자열 메소드중에 split() 메소드는 결과값을 list으로 반환합니다.

문자열의 join() 메소드는 list의 데이터를 하나의 문자열로 만들어 줍니다.

위 2가지 특성을 활용하여 다음 문제를 풀어보겠습니다.

```
str_text = ""first line  
second line  
third line""
```

- 앞서 여러 줄 문자열을 다음과 같이 출력합니다.
first line:second line:third line
- 앞서 여러 줄 문자열에서 두 번째 줄 첫 번째 단어를 second 출력
- 단어 사이를 : 문자로 연결하여 다음과 같이 출력
first:line:second:line:third:line

I 튜플

- tuple은 불변 리스트라고 불립니다. 곧 한번 만들면 변경이 안됩니다.
- list는 값을 넣거나 빼거나 변경이 자유로웠지만, tuple은 안됩니다.
- list처럼 값이 순서대로 저장되어 있고, 인덱스 값으로 데이터에 접근이 가능합니다.
- 형식
 - tuple 이름 = (원소_1, 원소_2, , 원소 N)
 - tuple은 소괄호를 사용하여 생성됩니다.
- 왜 사용할까요?
 - 데이터 바뀔 일이 없거나 바뀌면 안되는 경우 사용합니다.
 - 데이터가 의도치 않게 바뀔 가능성이 있다면 tuple로 사전에 예방합니다.

I 튜플

- 아래는 tuple 생성 예제입니다.
- tuple은 데이터 변경을 허용하지 않습니다.

```
days_of_the_week = ('일요일', '월요일', '화요일', '수요일', '목요일', '금요일', '토요일')
```

```
print (type(days_of_the_week))
```

```
<class 'tuple'>
```

```
days_of_the_week[0] = '이번주 일요일'
```

```
-----  
TypeError Traceback (most recent call last)  
<ipython-input-147-83ef49a797ec> in <module>  
----> 1 days_of_the_week[0] = '이번주 일요일'
```

```
TypeError: 'tuple' object does not support item assignment
```

I 튜플

- list와 tuple의 차이점에 대해서 알아보겠습니다.
- 공통점
 - 임의의 객체를 저장한다
 - 시퀀스 자료형이다
- 차이점
 - 변경 불가능
 - 메소드를 갖지 않는다
 - 포맷 문자열 지원, 함수 호출시 가변 인수 지원 기능 등이 추가로 있습니다.
- 일반적인 용도 차이
 - 데이터가 변경될 경우 – 리스트
 - 변경되지 않을 데이터를 표현할 경우 – 튜플
- 상호 변환 가능
 - list(), tuple()

I 튜플

- 패킹과 언패킹
 - 한 데이터에서 여러 개의 데이터를 넣는 것을 패킹(packing)이라고 합니다.
 - 패킹과 반대로 한 데이터에서 데이터를 각각 꺼내 오는 것을 언패킹(unpacking)이라고 합니다.
 - 언패킹시에 전달 받아야 할 변수의 개수가 맞지 않는다면 오류가 발생합니다.

```
x, y = t
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-173-757d34c4b6b9> in <module>
----> 1 x, y = t

ValueError: too many values to unpack (expected 2)
```

- 리스트도 언패킹은 지원합니다.

```
test = [1,2,3]
```

```
a, b, c = test
```

```
t = 1, 2, "Hello"
```

```
t
```

```
(1, 2, 'Hello')
```

```
x, y, z = t
```

```
x
```

```
1
```

```
y
```

```
2
```

```
z
```

```
'Hello'
```

I 튜플

- 추가적으로 tuple에 대한 설명을 하겠습니다.
 - 언패킹시에 좌변의 변수중 하나에 *를 붙이면 언패킹되고 남은 데이터들이 리스트 형식으로 저장됩니다.
 - 예제를 통해서 살펴보겠습니다.

가장 왼쪽 변수 a1에 *가 붙여 있기 때문에 a2, a3, a4에 데이터를 할당하고 남은 데이터를 a1에 할당합니다.

a2 : 6, a3 : 'test1', a4: 'test2'

a1에 데이터 하나를 할당하고 두번째 변수 a2에 *가 붙여 있기 때문에 a3, a4를 할당하고 남은 데이터 전체를 할당합니다.

a1에 데이터 하나, a2에도 데이터 하나, a3에도 데이터 하나 마지막 a4에 *가 붙여 있기 때문에 남은 데이터를 할당합니다.

```
t2 = 1, 2, 3, 4, 5, 6, 'test1', 'test2'
```

```
*a1, a2, a3, a4 = t2
```

```
a1
```

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

```
a1, *a2, a3, a4 = t2
```

```
a2
```

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

```
a1, a2, a3, *a4 = t2
```

```
a4
```

```
[4, 5, 6, 'test1', 'test2']
```

I 튜플

- for 문 표현식에서 for 변수에 list, tuple의 데이터를 언패킹되어 전달할 수 있습니다.
- 아래 예제를 통해서 확인해보겠습니다.

```
stock3 = [['삼성전자', '45300'], ['엘지전자', '73900']]  
for name, money in stock3:  
    print(name, money)
```

삼성전자 45300
엘지전자 73900

- 중첩된 list 안의 list의 값이 name, money에 언패킹되어 전달됩니다.
- tuple로 변경해도 동일한 결과가 나옵니다.

| 튜플

- 튜플의 존재의 이유
 - 어떤 연산을 더 효율적으로 만들어 주고, 또 어떤 연산을 안전하게 만들어 주기 때문입니다.
 - 한번 생성되고 나면 변경 할 수 없기 때문에 튜플은 항목을 추가/삭제 할 수 있는 자료형이 아니라 여러 부분으로 이루어진 단일 객체라고 이해하는 것이 편할 것입니다.
 - 개발자들은 List가 더 유연성이 있기 때문에 튜플을 완전히 무시하고 리스트만 사용하려는 경향이 있고, 리스트가 튜플에 비해 컴퓨터 메모리 사용이 많다는 사실만 참고.

```
tuple_name = ('삼성전자', '엘지전자', '삼성SDS')
list_name = ['삼성전자', '엘지전자', '삼성SDS']
print(tuple_name.__sizeof__())
print(list_name.__sizeof__())
```

48

64

I 사전(dictionary)

- Dict은 key-value 쌍으로 데이터를 보관하는 유형입니다.
 - Key – Value을 원소라고 부릅니다.
 - Dict은 중괄호 안에 쉼표로 구분하여 원소를 입력하여 만듭니다.
- Dict_이름 = { Key_1 : value_1 , Key_2 : value_2 }
- 빈 dict는 'dict_이름 = {}' 중괄호를 사용하면 만들 수 있습니다.
- 내부적으로 해시 기법을 사용하기 때문에 검색 속도가 빠릅니다.
- Key 값은 중복을 허용하지 않습니다.

```
first = {'Python' : '서찬웅', '패스트': '캠퍼스'}  
          _____ _____  
          key   value  key   value  
first  
  
{'Python': '서찬웅', '패스트': '캠퍼스'}
```

I 사전(dictionary)

- Dict에 새로운 원소를 추가하는 방법을 알아 보겠습니다.

```
stock_dict = {'삼성전자' : '45300', '엘지전자' : '73900'}
```

```
stock_dict
```

```
{'삼성전자': '45300', '엘지전자': '73900'}
```

```
stock_dict['삼성sds'] = '22400'
```

```
stock_dict
```

```
{'삼성전자': '45300', '엘지전자': '73900', '삼성sds': '22400'}
```

- 기존에 생성된 dict에 [key] = value 형식으로 입력하면 새로운 원소가 추가됩니다.
- 원소를 삭제하고 싶다면 del dict[key] 실행하면 됩니다.

```
del stock_dict['삼성전자']
```

```
stock_dict
```

```
{'엘지전자': '73900', '삼성sds': '22400'}
```

I 사전(dictionary)

- Dict에 원소가 존재하는지 확인하는 방법에 대해서 알아보겠습니다.

```
'엘지전자' in stock_dict
```

```
True
```

```
'삼성전자' in stock_dict
```

바로 전에 del를 이용해서 삼성전자를 삭제했기 때문에 False

```
False
```

- 시퀀스형 자료형에서 사용했던 in를 사용하면 dict안에 해당 key 값이 존재하는지 확인할 수 있습니다.

I 사전(dictionary)

- 이중이상으로 중첩된 dict에 대해서 알아보겠습니다.

```
stock_dict2 = {'삼성전자' : {'고가' : 46000, '저가': 45250, '시가' : 45750}, '엘지전자' : {'고가': 74900, '저가': 73600, '시가':74600}}
```

```
stock_dict2
```

```
{'삼성전자': {'고가': 46000, '저가': 45250, '시가': 45750},
 '엘지전자': {'고가': 74900, '저가': 73600, '시가': 74600}}
```

- Dict의 값은 데이터 유형에 상관없이 사용할 수 있습니다.
- 예제처럼 dict안에 dict을 넣을수도 있습니다.
- 이중, 혹은 그 이상으로 된 dict에서 value 값을 출력해 보겠습니다.
- Stock_dict2의 첫번째 Key '엘지전자'로 접근하고 두 번째 key인 '시가'로 접근하여 해당 value를 출력

```
stock_dict2['엘지전자']['시가']
```

```
74600
```

I 사전(dictionary)

- Dict의 메소드 keys(), values(), items()를 사용하여 데이터를 출력하는 방법에 대해서 알아보겠습니다.
 - keys() : 키에 대한 View를 반환
 - values() : 값에 대한 View를 반환
 - items() : (key, values)의 View를 반환
 - view는 dict 항목들을 동적으로 볼 수 있는 객체를 의미합니다.

```
stock_dict2.keys()
```

```
dict_keys(['삼성전자', '엘지전자'])
```

```
stock_dict2.values()
```

```
dict_values([{'고가': 46000, '저가': 45250, '시가': 45750}, {'고가': 74900, '저가': 73600, '시가': 74600}])
```

```
stock_dict2.items()
```

```
dict_items([('삼성전자', {'고가': 46000, '저가': 45250, '시가': 45750}), ('엘지전자', {'고가': 74900, '저가': 73600, '시가': 74600}))
```

- items() 메소드를 사용하여 데이터를 출력해보겠습니다.

```
for key, value in stock_dict2.items():
    print ("{} 주식현황 ".format(key))
    for key2, value2 in value.items():
        print ("{} = {}".format(key2, value2))
```

I 사전(dictionary)

- dict의 정렬을 해보겠습니다.
 - sorted() 함수를 사용하면 쉽게 정렬을 할 수가 있습니다.

```
dict_sorted = {'f' : 10, 'b' : 5, 'a' : 15, 'd' : 1}
```

```
sorted(dict_sorted, key=lambda x : dict_sorted[x])
```

value 기준 정렬

```
['d', 'b', 'f', 'a']
```

```
sorted(dict_sorted, key = lambda x : x)
```

key 기준 정렬

```
['a', 'b', 'd', 'f']
```

- 중첩 dict일 때 value 값으로 정렬하는 예제

```
stock_dict3 = {'삼성전자' : {'고가' : 46000, '저가' : 45250, '시가' : 45750},
               '엘지화학' : {'고가' : 371000, '저가' : 366500, '시가' : 370000},
               'SK텔레콤' : {'고가' : 245000, '저가' : 243000, '시가' : 244000},
               '엘지전자' : {'고가' : 74900, '저가' : 73600, '시가' : 74600}}
```

```
sorted(stock_dict3, key = lambda y : (stock_dict3[y]['시가']))
```

```
['삼성전자', '엘지전자', 'SK텔레콤', '엘지화학']
```

lambda는 함수편에서 공부할 내용입니다.
함수편에서 자세한 설명이 나옵니다.

I 사전(dictionary)

- dict의 메소드 리스트

- clear() – 항목 모두 삭제
- copy() – 사전 복사
- get(key[, x]) – 키가 없으면 x를 취한다
- setdefault(key[, x]) – 키가 없으면 x를 키에 설정한다
- update(D) – D사전의 내용으로 갱신
- popitem() – (키, 값) 을 리턴하고 항목 제거
- pop(key) - key 항목의 값을 반환하고 사전에서 제거

- 사전 내장(dictionary comprehension)

- 사전 내장은 중괄호 {}를 사용하여 키:값 형식으로 항목을 표현
- 사전 내장은 리스트 내장과 유사한 방식으로 동작하지만 사전을 만들어 낸다.

```
{w:k for k,w in [(1, 'one'), (2, 'two'), (3, 'three')]}  
{'one': 1, 'three': 3, 'two': 2}
```

| 참고

- 시퀀스(Sequence) 자료형 : 문자열, List, Tuple
- 자료형에 포함된 각 객체는 순서를 가지고 있으며, Index를 사용하여 접근 가능.
- Q : Index는 왜 1이 아닌 0부터 시작하는가?
- A : 컴퓨터는 이진수, 즉 비트(이진수)를 이용해 모든 것을 저장한다.
- 옛날에는 저장 매체의 가격이 매우 비싸서 하나의 비트라도 낭비하지 않기 위해서 메모리 위치와 List Index와 같은 것들도 0부터 시작하는 것.
- 가변과 불변 Sequence

가변 나열형	불변 나열형
list	str, tuple, range

I set(집합)

- set은 여러 값을 순서 없이 그리고 중복 없이 모아 놓은 자료형입니다.
- 순서가 없기 때문에 인덱스를 사용할수 없습니다.
- 반복 가능한(Iterable) 객체로부터 집합을 만들 수 있지만 모든 데이터가 집합의 원소로 사용 할수 있는 것은 아니다. Hashable 이면서 변경 불가능한 자료형만이 집합의 원소로 사용할 수 있습니다.
 - 정수, 실수, 복소수, 부울, 문자열, 튜플만이 집합에 저장할 수 있습니다.
 - list, set를 저장하려고 하면 TypeError가 발생합니다.

```
a = set()
b = {1, 2, 3}
a
set()
b
{1, 2, 3}
c = {}
type(c)
dict
```

빈 집합을 생성할 때 {}를 사용할 수 없는 이유와 빈 집합일 경우 출력이 중괄호 {} 대신 set() 함수인 이유는 중괄호{} 가 빈 사전으로 인식되기 때문이다.

```
a = {1,1,1,1,5,6,2,4,4,2,4,4,3,2,2,2,2,2,3,3,3,3,3}
```

```
a
{1, 2, 3, 4, 5, 6}
```

```
a[1]
```

```
TypeError
<ipython-input-89-8bc71255a22e> in <module>
      1 a[1]
-----
```

```
TypeError: 'set' object is not subscriptable
```

I set의 메소드

연산	동등한 표현	내용
s.update(t)	s = t	s와 t의 합집합을 s에 저장
s.intersection_update(t)	s &= t	s와 t의 교집합을 s에 저장
s.difference_update(t)	s -= t	s와 t의 차집합을 s에 저장
s.symmetric_difference_update(t)	s ^= t	s와 t의 배타집합을 s에 저장
s.add(x)		원소 x를 s에 추가
s.remove(x)		원소 x를 s에서 제거; 없으면 KeyError 예외 발생
s.discard(x)		원소 x가 있다면 s에서 제거
s.pop()		s에서 임의의 원소를 하나 리턴하고 집합에서는 제거; 빈 집합이면 KeyError 예외 발생
s.clear()		집합 s의 모든 원소 삭제

I set의 연산

- set은 수학의 집합하고 동일한 연산을 진행할 수 있습니다.
- union(합집합), intersection(교집합), difference(차집합), symmetric_difference(대칭 차집합)이 있습니다.

```
A = {1, 2, 3, 4, 5, 6}
B = {4, 5, 6, 7, 8, 9}
C = {4, 10}
```

A.union(B)

{1, 2, 3, 4, 5, 6, 7, 8, 9}

합집합 A | B 와 동일

A.intersection(B)

{4, 5, 6}

교집합 A & B

A.intersection(B, C)

{4}

인수가 2개 이상도 가능

A.difference(B)

{1, 2, 3}

차집합 A - B

A.symmetric_difference(B)

{1, 2, 3, 7, 8, 9}

대칭 차집합 A ^ B

✓ 결과가 반영되기를 원하면 update(), intersection_update(), difference_update(), symmetric_difference_update() 메서드를 사용하면 된다.

I set의 연산

- set은 중복을 허용하지 않기 때문에 이 성질을 이용하면 list 자료형에서 중복된 값을 제거할 수 있습니다.

```
a = [1,1,1,1,1,2,2,2,2,2,3,3,3,3,4,4,4,4,4,4,5,5,5,5,6,3,4,5,5,3,3,4,5,7,7,7]
```

```
set(a)
```

```
{1, 2, 3, 4, 5, 6, 7}
```

```
list(set(a))
```

```
[1, 2, 3, 4, 5, 6, 7]
```

| 정리

- 리스트(list) 생성
- 튜플(tuple) 생성
- 사전(dictionary) 생성
- 리스트 내장(list comprehension)
- 사전 내장(dict comprehension)
- 패킹, 언패킹
- 정렬
- 집합(set)

Chapter 08. 파이썬 프로그래밍 언어 파이썬 고급 주제들

| 이번시간에 배울 내용

1. 예외 처리
2. 맴 리듀스
3. 파일입출력

| 예외 처리에 대해서 알아보겠습니다.

- 예외처리는 프로그램이 무엇인가 잘못되었거나, 기대하지 않은 일이 발생 했을 때 알려주는 방법입니다.
- 모든 프로그램 언어는 예외 처리 구문이 존재합니다.
- 아래 예제는 list의 index 메소드를 사용할 경우 list 안에 값이 존재하지 않으면 오류(error)를 발생하는데 파이썬이 오류를 만나는 순간 오류의 내용을 출력하고 프로그램은 종료가 됩니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
stock.index('삼성SDS')
print ("이 문장은 실행이 되지 않습니다.")
```

```
ValueError                                Traceback (most recent call last)
<ipython-input-3-09468e9b332f> in <module>
      1 stock = ['삼성전자', '엘지전자', 'SK텔레콤']
----> 2 stock.index('삼성SDS')
      3 print ("이 문장은 실행이 되지 않습니다.")
```

ValueError: '삼성SDS' is not in list

- list의 index 메소드는 값이 없다면 Error가 발생한다. 삼성SDS가 없기 때문에 Error가 발생되고 print문은 실행이 되지 않는다.

I else 블록에 대해서 알아보겠습니다.

- Error가 발생 되었을 때 프로그램이 종료되지 않고 계속 실행이 될려면 프로그램에게 예외 발생 시에 어떻게 동작해야 할지를 알려주면 됩니다.
- 바로 그 구문이 try ~ except 블록입니다.

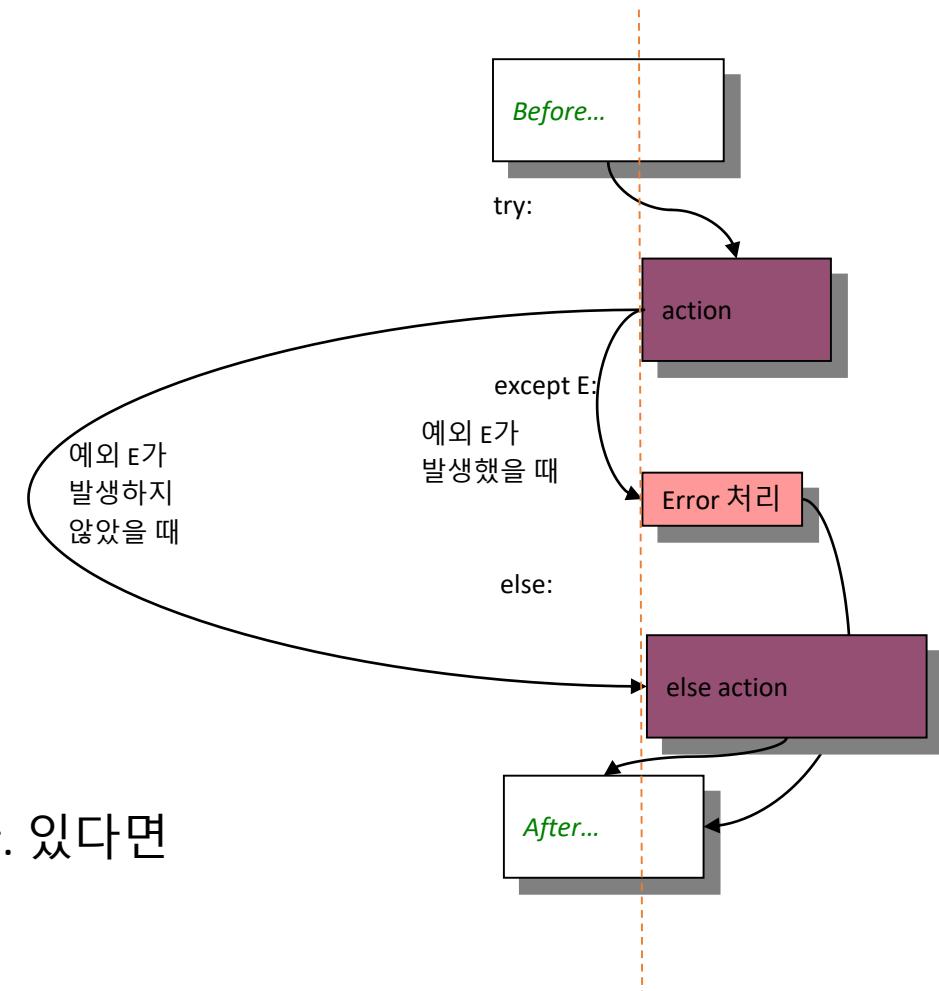
```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성SDS')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
```

삼성SDS가 없습니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성전자')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
```

원하는 데이터가 있습니다.

- else 블록은 옵션입니다. 즉 없을수도 있고 있을수도 있습니다. 있다면 try 블록에서 Error가 발생되지 않을 때 실행이 됩니다.



I finally 블록에 대해서 알아보겠습니다.

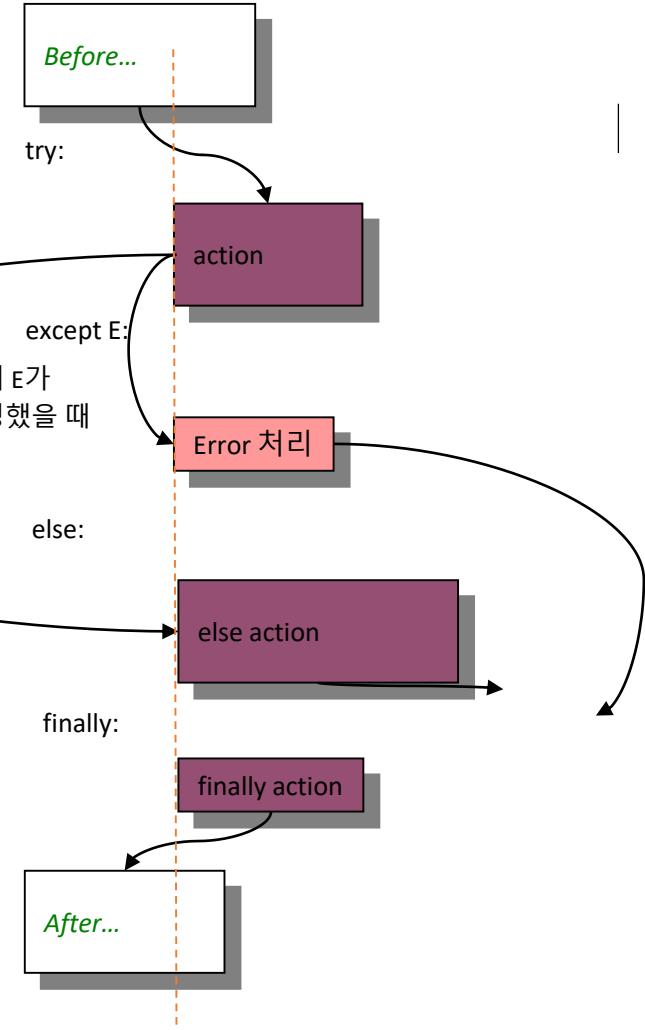
- finally 절은 예외 발생 여부에 상관없이 무조건 실행이 됩니다.
- 아래 예제에서 확인해보겠습니다.

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성SDS')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
finally:
    print ("무조건 실행")
```

삼성SDS가 없습니다.
무조건 실행

```
stock = ['삼성전자', '엘지전자', 'SK텔레콤']
try:
    stock.index('삼성전자')
except:
    print ("삼성SDS가 없습니다.")
else:
    print ("원하는 데이터가 있습니다.")
finally:
    print ("무조건 실행")
```

원하는 데이터가 있습니다.
무조건 실행



- finally 블록도 else블록과 마찬가지로 있을 수도 없을 수도 있습니다.
 - 단 있다면 Error 발생 여부와 무조건 실행이 됩니다.

I Error의 종류 및 상황별 예외 처리 방법에 대해서 알아보겠습니다.

- Error의 종류는 아래와 같이 여러 종류가 존재합니다.
 - SyntaxError : 파이썬 문법을 잘못 사용해서 발생하는 Error입니다.
 - AttributeError : 모듈의 없는 속성(메소드)를 호출 했을 때 발생합니다.
 - IndexError : 해당 시퀀스에 인덱스 범위를 잘못 지정하였을 때 발생합니다.
 - KeyError : 집합(set)과 사전(dict)에서 발생합니다.
 - 집합에서는 없는 원소를 remove()하면 발생
 - 사전에는 없는 key 값을 호출하면 발생
 - ValueError : 어떤 값을 가져올 수 없는 경우 발생
 - FileNotFoundError : 존재하지 않은 파일을 open() 할 때 발생합니다.
 - TypeError : 자료형에 적합하지 않은 연산을 할 경우 발생
- 앞에서는 try ~ except 블록에서 except만을 사용하여 Error 처리를 진행하였습니다. except만 사용할 경우는 다양한 종류의 Error를 모두 except 블록에서 처리하겠다 의미입니다.
- 만약 Error 종류별로 상황별 처리를 하고 싶다면 except SyntaxError, except AttributeError... 사용하시면 됩니다.

I Error의 종류 및 상황별 예외 처리 방법에 대해서 알아보겠습니다.

```
try:  
    5 / 0  
except NameError:  
    print ("NameError가 발생")  
except IndexError:  
    print ("IndexError가 발생")  
except ZeroDivisionError:  
    print ("ZeroDivisionError 발생")
```

ZeroDivisionError 발생

```
try:  
    stock[3]  
except NameError:  
    print ("NameError가 발생")  
except IndexError:  
    print ("IndexError가 발생")  
except ZeroDivisionError:  
    print ('ZeroDivisionError 발생')
```

IndexError가 발생

Error의 종류에 따라서 해당 Error를 처리할 수 있다

I Error 구체적인 내용을 확인하는 방법을 알아보겠습니다.

- 예외가 발생하였을 때 발생된 Error의 내용을 확인하고 싶다면 except Exception as e 구문을 사용하시면 됩니다.

```
try:  
    5 / 0  
except Exception as e:  
    print (e)
```

division by zero

- except 옆에 Error 종류를 적는 위치에 Exception만 적어도 except만 적는거와 동일한 결과가 발생됩니다.
 - except Exception:
 - except:
- except Error as e 표현식을 사용하면 Error의 구체적인 내용을 확인할 수 있습니다.
- 위의 예제처럼 except Exception as e 라고 표현하면 전체 Error에 대해서 해당되고 e 부분에는 구체적인 오류 메시지가 들어갑니다. 위의 예제에서 ZeroDivisionError만 처리하고 싶다면 except ZeroDivisionError as e :
 - as 키워드는 alias를 만들어주는 키워드이기 때문에 e 대신 사용자가 사용하고 싶은 단어를 쓰셔도 됩니다.

I pass에 대해서 알아보겠습니다.

- 아래 그림은 try ~ except ~ finally 전체 순서도입니다.
- 여기서 만약 except 구문에 아무 작업도 하고 싶지 않다면 어떻게 해야 할까요?
- 그럴 때 사용하는게 pass 키워드입니다.
- pass 키워드를 만나는 순간 아무 작업도 하지 않고 키워드 그대로 pass 합니다.
- try ~ except 구문 뿐만 아니라 for문에서도 자주 사용됩니다.

```
: for x in range(1,10):
    if x == 2:
        pass
        print ("After")
    print ("hmm")
```

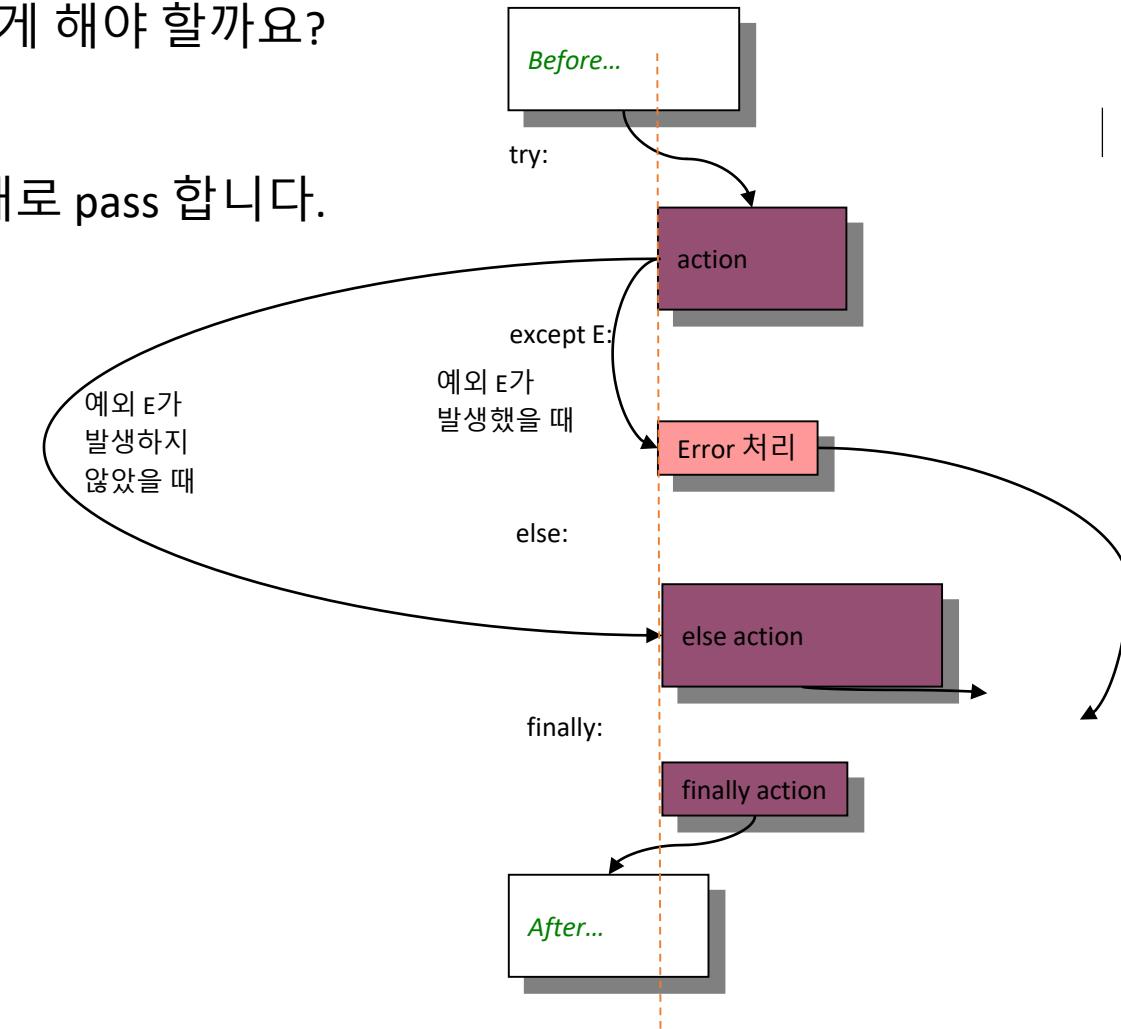
pass를 만나고 밑의 코드가 실행

hmm
After
hmm
hmm
hmm
hmm
hmm
hmm
hmm

```
: for x in range(1,10):
    if x == 2:
        continue
        print ("After")
    print ("hmm")
```

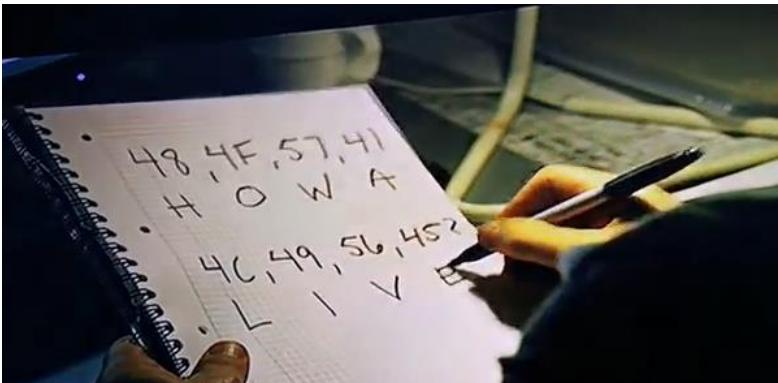
continue를 만나고 다음 for문 실행

hmm
hmm
hmm
hmm
hmm
hmm
hmm
hmm



I 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 문자를 사용할 때 대체로 아스키 문자 집합을 기본으로 사용합니다.
- 아스키 코드는 우리가 일반적으로 사용하는 각각의 문자에 0 ~ 127까지 붙여준 번호를 말합니다.
- 문자가 파일에 저장될 때는 문자에 대응되는 아스키 코드 값의 이진 데이터로 변환되어 저장됩니다.
- 문자를 어떤 규칙에 따라서 이진화(Binary: 0,1로 표현)하여 저장하는 것을 인코딩(encoding)이라고 합니다.
- 그리고 다시 문자로 바꾸는 작업을 디코딩(decoding)이라고 합니다.



```
Matt_Damon = "HOWALIVE".encode()

for idx, x in enumerate(Matt_Damon):
    print ("{:02x}".format(Matt_Damon[idx]))
```

이진 데이터(byte)로 변환

| 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 파이썬 3의 문자열은 유니코드이고, 유니코드는 세계적으로 사용하는 문자 집합을 하나로 모은 것입니다.
- 31비트로 표현되는 문자 세트이며, 온전히 한 문자를 표현하려면 4byte가 필요합니다.
- 파이썬의 문자열은 다수의 byte를 하나의 문자로 해석하는 유니코드 문자의 모임입니다. `"\ud30c\uc774\uc36c"`
- 참고로 '\u'를 사용하여 유니코드 문자를 지정할 수 있습니다. `"\u0048\u004f\u0057\u0041\u004c\u0049\u0056\u0045"`
`'HOWALIVE'`
- 유니코드는 4byte로 표현하는데 1byte, 2byte로 표현 할 수 있는 문자를 4byte로 표현을 하면 공간을 낭비 시킵니다. 또 이로 인하여 호환성 문제, 바이트 저장 순서등 문제가 발생됩니다.
- 따라서 호환성이 있으며 메모리를 절약할 수 있는 인코딩을 사용하는게 좋습니다. 이 때 사용하는 인코딩 기법인 UTF-8입니다.
- encoding을 사용하면 bytes 자료형으로 변환됩니다. 바이트는 문자열과 같이 변경 불가능 자료형이며, 문자열이 지원하는 대부분의 메소드를 지원합니다. 하지만 bytes와 문자열 간의 직접적인 연산은 허용되지 않습니다.

I 아스키코드, 유니코드, 인코딩, 디코딩, 바이트에 대해서 알아보기

- 아래 화면처럼 byte와 문자열간의 연산은 허용되지 않습니다.

```
In [158]: Matt_Damon + "like Apple"
Traceback (most recent call last):

  File "<ipython-input-158-157caebd1433>", line 1, in <module>
    Matt_Damon + "like Apple"

TypeError: can't concat bytes to str
```

- UTF방식은 UTF-8과 UTF-16이 있습니다.(나머지 방식은 존재하지만 거의 사용하지 않습니다.)
- UTF-8은 가장 널리 쓰이는 유니코드 인코딩인데 아스키 코드는 1byte로 인코딩되고 한글의 경우 3byte로 인코딩 됩니다. (문자의 특성에 따라서 용량이 동적으로 변화하기 때문에 가변길이 인코딩이라고 하기도 합니다.)

```
"파이썬".encode('utf8')
b'\xed\x8c\x8c\xec\x9d\xb4\xec\x8d\xac'
```

- 하지만 UTF-8로 저장된 파일은 BOM(Byte Order Mark) 정보가 없기 때문에 엑셀에서 파일을 오픈하게 되면 아래와 같이 나옵니다. (편집기에서 정상적으로 출력됩니다.)

```
f = open("gggg.csv", "w", encoding="utf8")
f.write("파이썬")
f.close()
```

I utf-8-sig로 저장하기

- BOM 정보가 추가된 utf-8-sig으로 인코딩하면 아래처럼 정상적으로 출력이 됩니다.

A	B
1	파이썬
2	

- 파일을 csv로 저장한다면 utf-8-sig를 사용하세요.
- BOM 표시인 FF FE가 추가된 것을 확인할수 있습니다.
- (참고 FF, FE은 Little Endian 방식으로 되어 있다는 의미입니다. 그냥 참고만...)
- 윈도우는 기본적으로 'cp949' 문자 집합을 사용합니다.
- 이메일 전송하고 한글이 깨지시는 분들은 cp949로 해보시기 바랍니다.

I 파일 읽기와 쓰기

- 프로그램이 생성한 데이터를 계속 유지하고 싶다면 데이터를 보관할 방법이 필요합니다.
- 반대로 보관된 데이터를 읽어오는 기술도 필요합니다.
- 키보드 입력을 읽어 화면에 출력했던 것처럼 파일로부터 입력을 받거나 출력하는 것 역시 가능합니다.
- 형식

- `open(파일 경로, mode, encoding='')`
- 파일 경로는 파일 이름을 포함하며 절대 경로와 상대 경로 모두 사용 가능
- `open` 함수의 mode

모드	설명
r	읽기 전용(기본 값)
w	쓰기 전용. 기존 내용 삭제
x	새 파일을 쓰기 전용 열기
a	기존 파일이 있는 경우 맨 뒤에 추가
b	이진 모드
t	텍스트 모드(기본 값)
+	업데이트 용도로 열기

- 파일은 `open()` 함수를 사용하여 파일을 읽고, `write()` 함수를 사용하여 기록합니다.
- `close()` 함수를 이용하여 파일을 닫아줘야 기록이 완성됩니다.

I 절대 경로와 상대 경로 알기

- 절대 경로
 - 파일 시스템의 Root로부터 시작하는 전체 경로를 의미합니다.
 - C:\test 라고 표시하면 절대 경로입니다.
- 상대 경로
 - 현재 작업 폴더에서 시작하는 경로
 - 파일 이름도 경로에 해당합니다.
 - 현 위치에서 .\test라고 하면 상대경로입니다.
- 현재 경로 구하기 (표준 라이브러리 os 사용)
 - os.getcwd()를 사용하여 현재 경로를 구할 수 있다.
- 현재 경로 변경하기
 - os.chdir()를 사용하여 경로를 변경할 수 있다.

I 파일의 메소드 알아보기

- `read()` 메소드를 사용하여 전체 txt파일을 읽어옵니다.
- `read()` 메소드를 사용하여 한번 읽어 들인 파일은 다시 실행해도 값을 읽어 올수 없습니다.
- 그 이유는 파일의 읽은 위치가 가장 마지막으로 설정되어 있기 때문입니다.
- `tell()` 메소드를 사용하여 파일에서 읽은 위치를 알수가 있습니다.
- `seek()` 메소드를 통해서 파일의 위치를 변경할 수 있습니다.
- `read()` 메소드를 사용하여 읽은 파일의 위치는 마지막에 위치, 다시 `read()`를 호출해도 값이 나타나지 않습니다.
- `seek(0)`으로 하여 파일의 위치를 0번으로 이동하여 다시 `read()`를 사용하면 그 위치부터 읽어 올 수 있습니다.
- `readline()` 메소드를 사용하면 한줄씩 읽어 올 수 있습니다.
- `readlines()` 메소드를 사용하면 라인 단위로 전체 파일을 읽어올 수 있습니다.
- `readlines()`의 반환값은 `list` 형식으로 반환합니다.
- `readlines()` 메소드는 파일의 내용을 메모리에 담아서 사용하기 때문에 대용량 파일인 경우 시스템이 멈출수도 있습니다.
- `with` 구문을 사용하면 메모리를 올려서 사용하는 방식이 아니라 대용량 파일도 처리할 수 있습니다.

| 정리

- 예외처리
- 맵 리듀스
- 파일입출력

Chapter 09. 파이썬 프로그래밍 언어

함수와 모듈

| 이번 시간에 배울 내용

1. 사용자 함수 작성 방법
2. 함수 인자값 전달
3. 지역변수, 전역변수
4. 람다함수
5. 모듈이란
6. 함수를 모듈로 만들기

| 함수란?

- 함수의 정의
 - 컴퓨터 프로그래밍에서 사용하는 개념 중에 같은 반복은 하지 말자라는 의미로 DRY(Don't Repeat Yourself)라는 말이 있다고 합니다.
 - 특정 작업을 수행 할 때마다 코드를 여러 번 작성하는 대신 해당 코드를 가진 함수를 만들어 호출합니다.
- 함수의 생성 규칙
 - 함수는 def 키워드 뒤에 함수를 식별할수 있는 이름을 적어서 생성합니다. 함수 이름 뒤에는 항상 괄호가 붙습니다.
 - def 함수이름():
 - () 안에 인자값을 받는다면 괄호 안에 인자 이름을 넣고 여러 개일 경우 쉼표로 분리합니다.

I 함수 예제

- 함수의 생성 예제
 - 아래 코드를 작성하고 실행하면 아무것도 출력이 되지 않습니다. 정의와 호출은 별개의 이야기입니다.

```
def say():
    print ("안녕하세요")
```

<함수의 정의>

say()

안녕하세요

<함수의 호출>

- 함수를 호출 할 때는 항상 소괄호를 붙어야 합니다.
- 함수 사용은 함수 생성 뒤에 사용할 수 있습니다. 함수를 생성 전에 호출하면 다음과 같은 오류가 발생합니다.

say2()

NameError Traceback (most recent call last)
 <ipython-input-3-88b0a69d2425> in <module>
 ----> 1 say2()

NameError: name 'say2' is not defined

I 함수의 인자값(매개변수)

- 함수는 인자(매개변수)를 전달 받아서 사용할 수 있습니다.
- 인자는 함수 안에서 사용할 수 있는 변수 이름입니다. (지역변수)

```
def say2(name):
    print ("{} 안녕하세요".format(name))
```

```
say2('수강생님')
```

수강생님 안녕하세요

- name은 say2 사용자 함수 안에서 사용하는 변수 이름입니다.
- 사용자 함수 정의할 때 인자를 정의하면 함수 호출 때는 반드시 인자값을 전달해야 합니다. 그렇지 않으면 아래와 같은 오류가 발생합니다.

```
say2()
```

```
TypeError                                 Traceback (most recent call last)
<ipython-input-8-88b0a69d2425> in <module>
      1 say2()
```

TypeError: say2() missing 1 required positional argument: 'name'

I 함수에 여러 개의 인자값을 전달하기

- 함수를 정의할 때 인자 값을 여러 개를 정의할 수 있습니다.
- 정의할 때 소괄호() 안에 여러 인자 이름을 콤마(,)로 구분하여 입력하시면 됩니다.
- 호출 할 때도 인자 값을 콤마(,)로 분리해서 호출하면 같은 위치에 있는 값들이 매핑되어 실행이 됩니다.

```
def stock(name, start):
    print ("{}의 시가는 {}입니다.".format(name, start))
```

```
stock('삼성전자', '43000')
```

삼성전자의 시가는 43000입니다.

- name = 삼성전자, start = 43000이 대입되어 함수가 실행이 되었습니다.
- 인자 이름을 각각 지정해서 함수에 전달하면 이름에 맞게 대입되어 값이 전달이 됩니다.

```
stock(start=43000, name='삼성전자')
```

삼성전자의 시가는 43000입니다.

- 인자값 이름을 지정하여 전달할 때는 오타등의 이유로 이름을 잘못 부르면 오류가 발생합니다.
- 하나의 인자에 여러 개의 값을 전달 할 수도 있습니다. 즉 가변인수를 지원합니다. 인자 값에 * 기호를 붙입니다.

I 함수의 인자 값을 기본값을 설정하기

- 인자의 기본값을 원할 경우 함수 정의할 때 원하는 인자에만 설정할 수 있습니다.

```
def stock2(start, name = '삼성전자'):  
    print ("{}의 시가는 {}입니다.".format(name, start))
```

```
stock2(43000, '삼성전자')
```

삼성전자의 시가는 43000입니다.

- Q : stock(name, start)으로 정의했는데, 왜 기본 값이 있는 함수에는 인자값의 순서가 변경되었을까요?
- A : 함수를 정의할 때 기본값이 없는 인자가 기본값이 있는 인자보다 무조건 앞에 나와야 합니다. 그렇지 않으면 오류가 발생합니다.

I 함수의 반환 값(return)

- 함수는 특정 작업을 수행하고 return 구문을 사용하여 처리된 데이터를 반환할 수 있습니다.
- 원하는 곳에서 어떤 데이터라도 반환할 수 있습니다.
- 그 역할을 하는 키워드가 return입니다. return을 호출하게 되면 값을 바로 반환하고, 그 함수는 종료가 됩니다.

```
def odd_or_even(number):
    if number % 2 == 0:
        return '짝수'
    else:
        return '홀수'
print ('이 프린트 문이 실행이 될까요?')
```

```
odd_or_even(15)
```

```
'홀수'
```

```
def odd_or_even2(number):
    if number % 2 == 0:
        return number, '짝수'
    else:
        return number, '홀수'
print ('이 프린트 문이 실행이 될까요?')
```

```
odd_or_even2(15)
```

```
(15, '홀수')
```

- return 값이 2개 이상일 때는 return 된 결과 값이 튜플로 구성되어 전달이 됩니다.
- 튜플로 구성된 값은 언패킹을 사용해서 각각의 변수에 입력할 수 있습니다.

```
numb, text = odd_or_even2(15)
```

```
numb
```

```
15
```

```
text
```

```
'홀수'
```

| 지역변수란?

- 변수가 사용되는 프로그램의 영역을 유효범위라고 합니다.
- 아래 calculateTax 함수의 매개변수로 선언된 price는 함수 내부에서만 사용하는 변수입니다.
- 함수 외부에서 호출하게 되면 정의되지 않은 값이라고 오류가 발생합니다.

```
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    return total
```

```
my_price = float(input("Enter a price : "))
```

Enter a price : 20000

```
total_price = calculateTax(my_price, 0.06)
```

```
print ("price = {}, Total = {}".format(my_price, total_price))
```

price = 20000.0, Total = 21200.0

```
print (price)
```

```
NameError                                 Traceback (most recent call last)
<ipython-input-58-3b154833d11b> in <module>()
      1 print (price)
```

NameError: name 'price' is not defined

I 전역변수

- 프로그램에서 정의된 변수명은 사용자 함수에서 변경하지 않은 이상 전역변수로 선언됩니다.
- 전역변수로 선언되면 프로그램 전체에서 사용이 가능합니다.

```
def calculateTax(price, tax_rate):
    total = price + (price * tax_rate)
    print ("your price {}".format(your_price))
    return total
```

your_price가 전역변수로 선언되어 프로그램 전체에서 사용 가능하다

```
your_price = 99999999
my_price = float(input("Enter a price : "))
```

Enter a price : 20000

```
total_price = calculateTax(my_price, 0.06)
```

your price 99999999

```
print ("price = {}, Total = {}".format(my_price, total_price))
```

price = 20000.0, Total = 21200.0

| 전역변수와 지역변수와 동일한 이름의 변수의 우선순위

- 프로그램이 생성할 때 프로그램 코드의 길이가 길어지고 나면 동일한 변수명을 전역, 지역에 사용할 때가 발생할 수 있습니다.
- 이럴 경우 꼭 기억하실 내용은 **지역변수가 전역변수보다 우선 순위가 높다**라는 점을 기억하세요.
- 함수 내부 곧 지역변수에서 동일한 이름의 전역변수를 사용하고 싶다면 global 키워드를 사용하면 됩니다.

```
def test():
    x = 1
    print (x, id(x))

x = 10
test()
print (x, id(x))
```

1 140712295306048
10 140712295306336

```
def test():
    global x
    x = 1
    print (x, id(x))

x = 10
test()
print (x, id(x))
```

1 140712295306048
1 140712295306048

| 람다함수

- 람다 함수는 이름 없는 한 줄짜리 함수입니다.
- 사용 방법은 lambda 변수명 : 표현식

```
def my_function(x):
    return x**2
```

```
my_function(5)
```

```
25
```

```
(lambda x : x**2)(5)
```

```
25
```

```
dict_sorted = {'f' : 10, 'b' : 5, 'a' : 15, 'd' : 1}
```

```
sorted(dict_sorted, key=lambda x : dict_sorted[x])
```

```
['d', 'b', 'f', 'a']
```

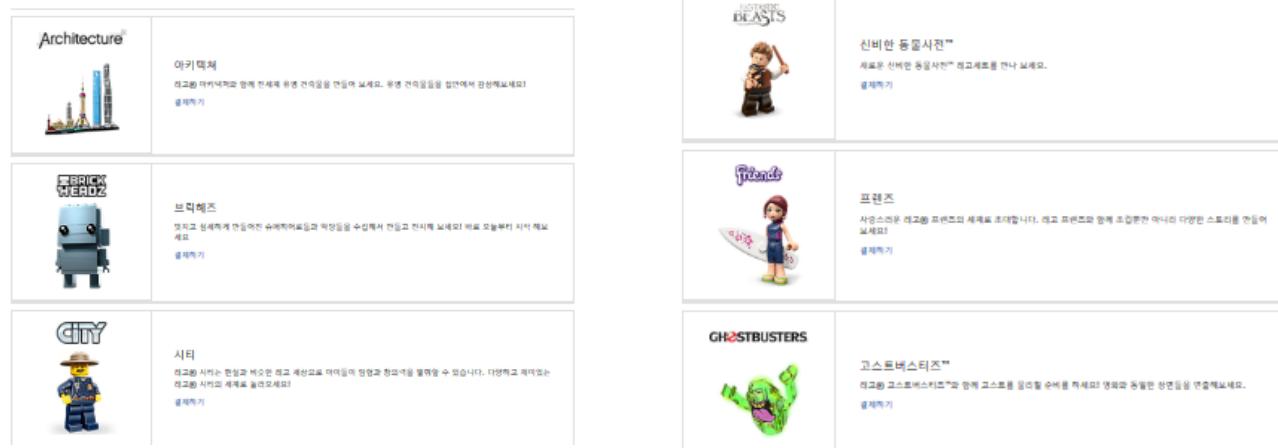
```
sorted(dict_sorted, key = lambda x : x)
```

```
['a', 'b', 'd', 'f']
```

구분	def로 정의되는 함수	lambda 함수
문/식	문(statement)	식(expression)
함수의 이름	def 다음에 지정된 이름으로 생성한 함수 객체를 치환한다	함수 객체만을 생성한다
몸체	한 개 이상의 문을 포함한다	하나의 식만 온다
리턴	return 문에 의해 명시적으로 리턴 값이 지정된다	식의 결과가 리턴된다
내부 변수 선언	지역 영역에 변수를 생성하고 사용하는 것이 가능하다	지역 영역에 변수를 생성하는 것이 가능하지 않다

I 모듈(module)

- 모듈은 서로 관련이 있는 프로그램 코드들을 모아놓은 파일입니다.
- 파이썬에서는 모듈은 커다란 프로그램을 구성하는 작은 조각을 의미합니다.
- 파이썬 프로그램으로 된 파일이거나 혹은 C언어로 만들어진 파이썬 확장 파일일 수도 있습니다.
- 파이썬을 레고라고 생각하면 모듈은 레고 시리즈라고 생각하시면 편할거 같습니다.
- 레고는 단순하지만 레고 상품마다 난이도도 다르고, 사용자가 원하는 상품만 구매하면 되듯이 파이썬도 모든 모듈을 다 사용할 필요 없이 원하는 모듈만 사용하면 됩니다.



<사용자가 원하는 시리즈만 구매할수 있는 레고사이트>
출처 : <https://shop.lego.com/ko-KR/category/themes>

I 모듈은 왜 사용할까?

- 모듈은 표준 모듈, 사용자 정의 모듈, Third Party 모듈이 존재합니다.
 - 표준 모듈은 파이썬 설치시 기본적으로 제공되는 모듈로 파이썬 기본 라이브러리라고도 합니다.
 - 사용자 정의 모듈은 사용자가 직접 만들어서 사용하는 모듈입니다.
 - Third Party 모듈은 Selenium같은 기본적으로 제공하는 모듈이 아닌 외부 제 3자가 만든 모듈입니다.
- 왜 모듈을 사용할까요?
 - 코드를 재 사용할 수 있습니다.
 - 파일의 크기가 더 작아져서 코드에서 원하는 것을 찾기가 쉽습니다.
 - 모듈은 시스템 구성 요소의 기본 단위입니다. 파이썬에서 모든 것을 모듈 단위로 분리해서 관리하고 있습니다. 하나의 시스템을 모듈 단위로 분리해서 설계하는 것은 여러 면에서 작업의 효율을 높일 수 있습니다.
 - 모듈은 별도의 Name Space를 제공합니다. 다른 모듈과 겹치지 않은 공간을 가지기 때문에 다른 모듈의 함수나 변수에 영향을 받지 않고 독립적인 작업을 할 수 있다.

I NameSpace

- A부터 F까지의 클래스가 있는 학교가 있습니다. 이 학교에는 신기하게도 A~F클래스에 마이클이라는 이름을 가진 학생이 존재한다고 합니다. 학교 전체에서 보면 6명의 마이클이 존재하는 거겠죠.
- 강당에 모든 학생들을 모아놓구 교장 선생님이 외칩니다. ‘마이클~~~ 자넨 퇴학이야’
- 6명의 마이클은 모두 깜놀할거 같습니다.
- 이렇게 교장이 전교생을 모아놓구 외치는 것을 **Global NameSpace**에 외치는 것과 같습니다.
- Global NameSpace에 외치면 위의 상황같이 교장이 원하는 한명을 지목하기 힘듭니다. 그래서 교장 선생님은 다시 외칩니다. ‘A 클래스의 마이클 퇴학이야~’
- 순간 A클래스의 마이클만 반응합니다. 나머지 마이클들은 안도의 한숨을 쉽니다.
- 이렇게 교장 선생님이 A클래스라고 구역을 분리했기 때문에 A클래스의 마이클만 해당되는 것입니다.
- 이를 **Local NameSpace**라고 합니다.

I NameSpace

- 다시 파이썬으로 돌아와서 파이썬에서 모듈을 가져오는 방법이 2가지 존재합니다.
 - `from <모듈> import <함수 or 클래스>`
 - `import 'A클래스'`
 - 이렇게 선언하게 되면 내가 원하는 마이클을 호출할때마다 `A클래스.마이클` 이라고 해야 합니다.
- `from A클래스 Import '마이클'`
 - A클래스의 있는 마이클만 호출하겠다는 의미입니다.
 - 이제부터 마이클을 호출하면 A클래스의 마이클만 대답을 할 것입니다.
 - '마이클'이라는 이름을 계속 부르면 나머지 5명에게 웬지 미안할거 같습니다. 그래서 '마이클'에게 '미남'이라는 별명을 붙여줍니다.
 - `from A클래스 import '마이클' as '미남'`
 - 만약 A클래스나 특정 클래스의 학생들을 모두 호출하고 싶다면 ...
 - `from A클래스 import *`

I 사용자 모듈 만들기

- 아래와 같이 my_module.py를 작성합니다.

The screenshot shows a code editor window titled "my_module.py" containing Python code. The code defines a module named "my_module" with two functions: "c_to_f" which converts Celsius to Fahrenheit, and "f_to_c" which converts Fahrenheit to Celsius. A docstring at the top explains the module's purpose. To the right is a file browser listing several Python modules:

파일	날짜	타입	크기
macpath.py	2019-03-26	PY 파일	6KB
mailbox.py	2019-03-26	PY 파일	77KB
mailcap.py	2019-03-26	PY 파일	8KB
mimetypes.py	2019-03-26	PY 파일	21KB
modulefinder.py	2019-03-26	PY 파일	23KB
my_module.py	2019-04-22	PY 파일	1KB
netrc.py	2019-03-26	PY 파일	6KB
nntplib.py	2019-03-26	PY 파일	43KB
ntpath.py	2019-03-26	PY 파일	22KB

- anaconda 설치 폴더의 Lib 밑에 모듈이 저장되어 있습니다. 해당 폴더에 my_module.py를 저장합니다.
- 저장 뒤에는 모듈을 호출 할수 있습니다.

```
import my_module
```

```
my_module.c_to_f(22)
```

```
71.6
```

```
from my_module import c_to_f
```

```
c_to_f(22)
```

```
71.6
```

| 정리

- 사용자 함수 만들기
- 인자 값 전달하기
- return 키워드
- 지역변수, 전역 변수
- 람다함수
- 모듈 및 from, import의 의미
- 사용자 모듈 만들기

Chapter 10. 파이썬 프로그래밍 언어 흐름 제어 (조건문과 반복문)

| 이번시간에 배울 내용

- 조건문 - if 문
- 반복문 - for 문
- continue, break
- range()함수
- 반복문 - while 문

| 조건문

어느 아내가 프로그래머 남편에게

아내 : 우유 하나 사와. 아, 달걀 있으면 6개 사와

남편은 우유를 6개 사왔다.

아내는 물었다.

아내 : 왜 우유를 6개나 사왔어!

남편: 달걀이 있길래 6개 사왔지

남편의 생각

IF 달걀 있는가?

우유 6개를 산다.

ELSE 달걀이 없다면 :

우유 하나 사온다.

I 조건문

- if문은 조건문에서 조건식이 True인지 False인지를 판단하여 문장을 실행하는데 조건식은 우리가 앞에서 배운 비교 연산자, 논리 연산자의 결과 값을 받아서 사용합니다.
 - if(만약) 조건이 참이라면, if 블록의 명령문을 실행
 - else(아니면) else블록의 명령문을 실행
 - 이 때 else 조건절은 생략이 가능

五題

if 조건식1: ← 조건식으로 끝난 문장은 :를 붙여야 한다.

statement1

elif 조건식 2: ↗ elif → else if의 줄인 글

elif → else if의 줄인 말

statement2

else:

statement3

I 조건문

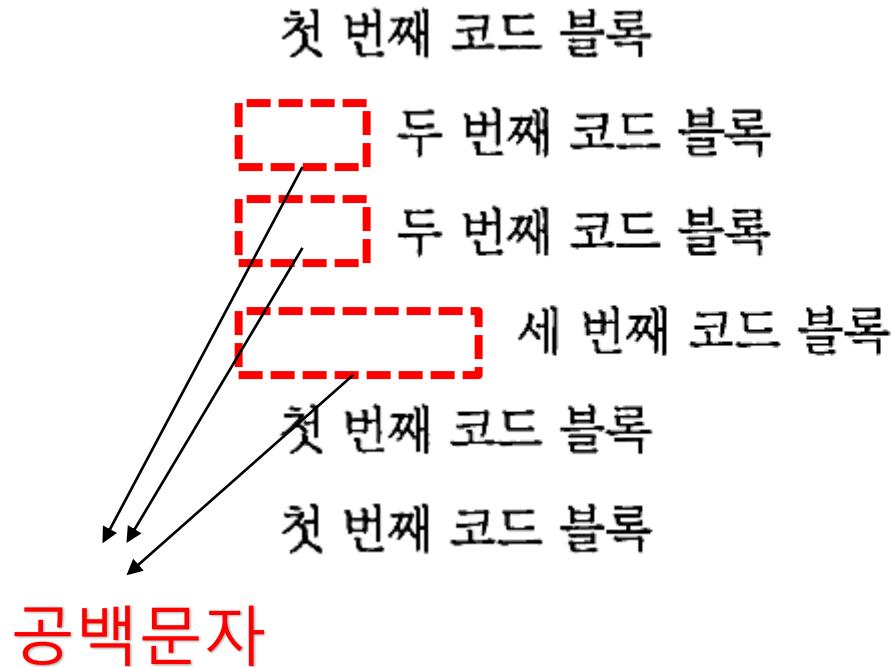
- if문의 예제를 보면서 자세히 알아보겠습니다.



- 35<40의 표현식만 본다면 오른쪽 그림처럼 True입니다.
- True 값이기 때문에 if 구문의 아래 표현식 코드가 실행이 되었습니다.
- print 함수 앞에 공백을 우리는 앞에서 배운 들여쓰기(indent block)라고 부릅니다.
- 파이썬에서는 같은 깊이만큼(공백 또는 탭) 있는 표현식은 모두 같은 코드 블록에 있다고 인식합니다.

I 조건문

- 일반적으로 4 개의 공백 문자를 사용하지만 강제는 아닙니다.
- 이전 설명처럼 프로그래머가 원한다면 변경할 수 있습니다.
- 단 모든 들여쓰기를 일관되게 작성해야 합니다.
- 공백 2 개를 들여쓰기로 사용했다면 같은 코드 블록은 공백 2 개로 들여쓰기 되어 있어야 합니다.



I 조건문

- 아래 예제의 결과를 예상해 보세요.

```
speed = 120
if speed >= 110:
    print ("과속입니다.")
print ("안전운행")
```

- if 문은 위의 예제처럼 독립적으로 사용될 수 있지만, else 문과 함께 사용할 수 있습니다.
- else 아래 들여쓰기로 써진 코드는 if문이 거짓일 때(False) 실행이 됩니다.
- else문에도 : 가 붙어 있는걸 유의하세요. : 다음 줄은 들여쓰기가 시작되고 하위 코드들입니다.

```
speed = 100
if speed >= 110:
    print ("과속입니다.")
else:
    print ("정상속도입니다.")
print ("안전운행")
```

I 조건문

- else 구문을 할때 다시 if 조건을 줄 수 있습니다.
- 사람의 언어로 표현하면 '아니라면 그럼 다른 조건을 제시할께 " 정도 될거 같습니다.
- elif는 else if를 줄인 단어로 역시 조건문이 True일 때 실행할 코드는 : 다음 줄, 즉 들여쓰기가 시작되는 블록

```
speed = 120
if 130 <= speed:
    print ("130km 과속입니다.")
elif 120 <= speed:
    print ("120km 과속입니다.")
elif 110 <= speed:
    print ("110km 과속입니다.")
else:
    print ("정상속도입니다.")
print ("안전문행")
```

- else 아래 코드는 최종적으로 모든 조건이 False일 때 실행됩니다.
- 만약 else 위에 조건문들중 참이 실행이 되면 해당 코드블록만 실행되고 조건문은 종료가 됩니다.
곧, else 표현식은 실행이 되지 않습니다.

I 조건문

- if 문을 사용해서 다음 문장을 코드로 표현하고 실행하세요.

- 문제

Distance라는 변수에는 상수 값이 들어갈 예정입니다.

이 distance가 3km이하로 작다면 “걸어가세요” 라고 말하고

3km보다 멀고 10km보다 작다면 “버스타세요”라고 말해야 합니다.

만약 10km이상 간다면 “택시 타세요”라고 답할수 있는 조건문을 만들어보세요.

그리고 조건문과 상관없이 마지막에 “수고하셨습니다.”라고 출력하세요.

I 반복문 - for 문

- for 문을 반복문이라고 합니다. 가장 많이 사용하며, 형식은 아래와 같습니다.

```
for <target> in <object>:
```

```
    <statement1>
```

```
else:
```

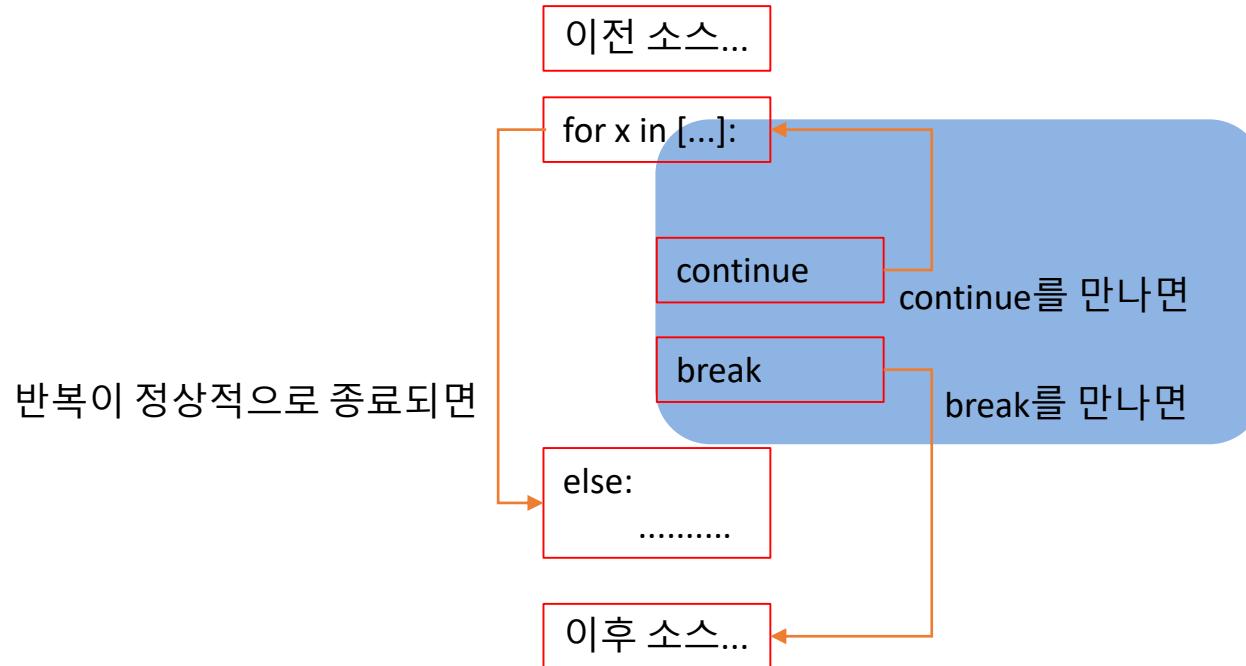
```
    <statement2>
```

조건문 뒤에는 : 를 잊지 말자

- <object>는 시퀀스형 데이터가 들어가야 합니다.
- <object>의 각 항목은 <target>에 치환되어 <statement1>이나 <statement2>를 실행합니다.
- 반복의 횟수는 <object>의 크기가 즉 시퀀스의 크기가 됩니다.
- for 문에서 사용할 수 있는 객체는 아래와 같습니다.
 - 문자열, 리스트, 튜플, 사전, range() , 반복이 가능한 객체

I 반복문 - for 문 - continue, break

- for 문의 내부 블록에서 continue를 만나면 시작 부분으로 이동, break를 만나면 for문의 내부 블록을 종료합니다.
- break, continue 표현식은 if 문과 함께 표현합니다.



I range() 함수

- 내장 함수인 range() 함수는 숫자로 이루어진 값을 만듭니다.
- for 문과 같이 사용합니다. range() 함수는 한 개에서 세 개까지의 인수를 받을 수 있습니다.
- range(x)와 같이 인수가 주어져 있을 경우, 0부터 인자값 x보다 작은 수까지의 리스트를 1간격으로 생성
- range(x, y)인 경우 x부터 y보다 작은 값(y-1)으로 데이터를 시퀀스하게 만듭니다.
- range(x, y, s)인 경우 range(x, y)와 같지만 간격을 s로 하여 값을 생성을 합니다.

```
for number in range(3):
    print (number)
```

0
1
2

```
for number in range(1,3):
    print (number)
```

1
2

```
for number in range(1,10,2):
    print (number)
```

1
3
5
7
9

I 반복문 - for 문 - continue, break

- 아래 예제를 보면서 결과를 예측해 봅시다.
- if 조건을 추가하여 조건식이 참이면 break, 혹은 continue 합니다.
- for문에서 else문은 for 문 블록이 강제로 종료(break) 되지 않아야 실행이 됩니다.

```

for i in range(1,13):
    if i % 2 == 0 and i % 3 == 0:
        print (i)
        break
        print ('if statement')

else:
    print ('complete')
    
```

```

for i in range(1,13):
    if i % 2 == 0 and i % 3 == 0:
        print (i)
        continue
        print ('if statement')

else:
    print ('complete')
    
```

I 문제

- 아래 for문은 1부터 100까지 정수를 출력하는 코드입니다.
- 아래 조건을 추가하여 만들어 보세요.

```
for x in range(1, 101):  
    print (x)
```

- 조건 : 2의 배수를 출력해 주세요. 단 5의 배수 일때는 출력을 하지 않습니다.

I while문

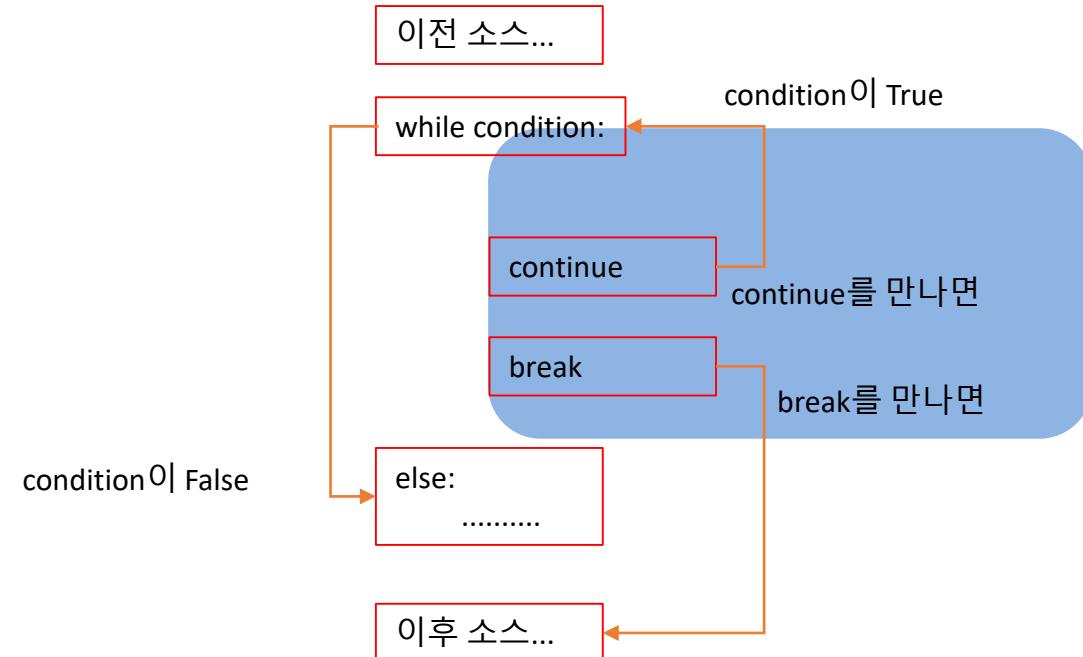
- for 문보다 일반적인 반복문입니다.
- 시작 부분의 <condition>을 검사해서 결과가 True 이면 내부 블록이 반복적으로 실행됩니다.
- else 블록은 <condition>이 False일 때 while문을 빠져나올 때 실행됩니다.
- break로 빠져나올 때는 else 블록은 실행되지 않습니다.

while <condition>[:]

 <statement>

else:

 <statement>



I while문

```
In [125]: a = 0
....: while a < 10:
....:     a = a + 1
....:     if a < 3:
....:         print (a)
....:         continue
....:     print ('continue')
....:     if a > 10:
....:         break
....:     else:
....:         print ('else block')
....: print ('done')
```

a에 +1 하여 다시 a에 대입

a가 3보다 작을 때 실행 print(a) 가 출력되고
continue를 만나 처음으로 이동

a가 10보다 크지 못하기 때문에 if문은 항상 False
break문은 실행되지 못함

정상적으로 종료되었기 때문에 else 구문이 실행

```
1
2
else block
done
```

while문과 같은 블록 위치를 같기 때문에 while문과 독립적으로 print('done') 실행

```
In [128]: a = 0
....: while a < 10:
....:     a = a + 1
....:     if a < 3:
....:         print (a)
....:         continue
....:     print ('continue')
....:     if a > 6:
....:         break
....:
....:     else:
....:         print ('else block')
....: print ('done')
```

a가 6보다 클 때 True가 되기 때문에 break문 실행 while 종료

정상적으로 종료가 되지 않았기 때문에 else는 실행되지 않는다.

```
1
2
done
```

I while문

- 참고

- effective Python 책에서는.... 아래와 같이 권고합니다.
 - 반복문 뒤에 else 블록을 사용하면 직관적이지 않고 혼동하기 쉬우니 사용하지 말 것.

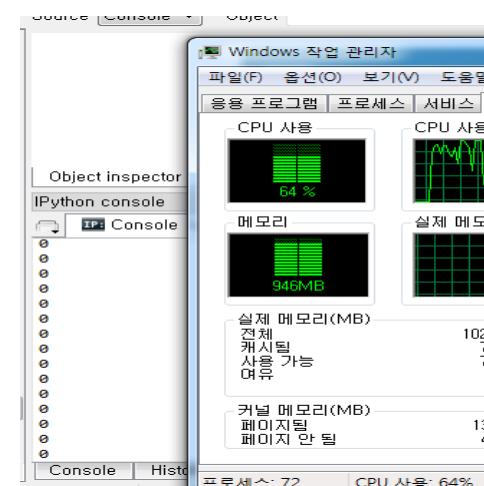
- for 문과 while 문의 차이점

- for문은 조건 및 집합의 순환을 위해서 반복 할 수 있습니다.
- while 문은 하고 싶은 어떤 종류의 반복적인 구성도 할 수 있습니다.
- 하지만 while문은 무한 루프를 일으킬 수 있기 때문에 보통은 for 문으로 많은 일을 하게 됩니다.

```
In [1]: i = 0

In [2]: while i < 5:
...:     print i
...:     i = i + 1
...:

0
1
2
3
4
```



| 실습

- 윤년(閏年)은 역법을 실제 태양년에 맞추기 위해 여분의 하루 또는 월(月)을 끼우는 해이다. 태양년은 정수의 하루로 나누어떨어지지 않고, 달의 공전주기와 지구의 공전주기는 다르기 때문에 태양력에서는 하루(윤일), 태음태양력에서는 한 달(윤달)을 적절한 시기에 끼워서 이를 보정한다. - Wikipedia
- 윤년의 규칙
 - 1. 연수가 4로 나누어 떨어지는 해는 윤년으로 한다.
 - (2004, 2008, 2012, 2016, 2020)
 - 2. 이 중에서 100으로 나누어 떨어지는 해는 평년으로 한다.
 - (1900, 2100, 2200, 2300)
 - 3. 그 중에서 400으로 나누어 떨어지는 해는 윤년으로 둔다.
 - 위의 3가지 규칙을 사용하여 1900년부터 2100년까지 윤년을 출력하시요.

| 정리

- if 문을 사용한 제어문에 대한 학습
- for 문과 while 문을 사용하여 반복문을 학습
- for 문과 while 문의 차이
- range() 함수를 사용한 데이터 생성
- continue, break 문을 통한 반복문 제어