# Python 프로그래밍

# 파이썬 개요

- 파이썬은 배우기 쉽고, 강력한 프로그래밍 언어
- 파이썬은 효율적인 고수준 데이터 구조를 갖춤
- 간단하지만 효과적인 객체 지향 프로그래밍 접근법 또한 갖춤
- 우아한 문법과 동적 타이핑, 그리고 인터프리팅 환경을 갖춘 파이썬은 다양한 분야, 다양한 플랫폼에서 사용될 수 있는 최적의 스크립팅, RAD(rapid application development - 빠른 프로그램 개발) 언어

# 파이썬 개요

**파이썬이라는 이름의 유래**

파이썬의 창시자 귀도 반 로섬(Guido van Rossum)이 BBC에서 방영되던 "Monty Python's Flying Circus"라는 TV 프로그램의 이름을 따서 지었음.

# 파이썬 특징

**단순함**

- 파이썬은 단순하고 최소화된 언어
- 잘 쓰여진 파이썬 프로그램을 읽는 것은 좀 딱딱하게 쓰여진 영어 문장을 읽는 것과 같음
- 이러한 프로그램 코드같지 않아 보이는 특성은 파이썬의 가장 강력한 특성들 중 하나
- 이로 인해 파이썬이라는 언어 자체보다 여러분이 풀고자 하는 문제에 더 쉽게 집중할 수 있음.

# 파이썬 특징

**배우기 쉬운 언어**

곧 알게 되시겠지만, 파이썬은 정말 배우기 쉬운 언어입니다. 위에서 이미 이야기했지만, 파이썬은 굉장히 쉬운 문법 체계를 갖고 있습니다.

# 파이썬 특징

## 자유, 오픈 소스 소프트웨어

파이썬은 FLOSS (Free/Libré and Open Source Software - 자유, 오픈 소스 소프트웨어)의 좋은 예입니다. 이것은 이 소프트웨어의 복사본을 마음대로 배포할 수 있고, 소스 코드가 공개되어 있어 언제든지 읽을 수 있으며, 필요한 부분을 고칠 수 있고, 새로운 자유 소프트웨어를 작성할 때 이 프로그램의 일부를 사용해도 된다는 것을 의미합니다.
FLOSS는 지식을 공유하는 공동체를 기반으로 하고 있습니다. 이것은 왜 파이썬이라는 언어가 이렇게 좋은 언어가 되었는지를 설명하는 좋은 이유입니다. 파이썬은 좀 더 나은 파이썬을 만들고자 하는 공동체의 노력에 의해 지속적으로 개선되고 있기 때문입니다.

# 파이썬 특징

**고수준 언어**
여러분이 파이썬으로 프로그램을 작성할 때, 메모리를 관리한다던가 하는 저수준의 세부적인사항들을 신경쓸 필요가 없다.

**이식성**
파이썬은 소스가 공개되어 있으므로, 여러 플랫폼을 지원하도록 수정되어 왔다. 따라서 여러분이 프로그램을 작성할 때 특정 플랫폼에서만 사용되는 몇몇 기능들을 사용하지 않으면, 작성한 모든 파이썬 프로그램은 어떤 수정 없이도 파이썬이 동작하는 모든 플랫폼 위에서 동작할 수 있음.

# 파이썬 특징

**인터프리터 언어**

컴파일러 언어인 C 혹은 C++로 작성된 프로그램은 컴파일러에 여러 옵션과 플래그를 주어 프로그래머가 작성한 소스 코드로부터 컴퓨터가 사용하는 언어 (0과 1로 구성된 바이너리 코드)로 번역하게 하는 과정을 거칩니다. 이렇게 번역된 프로그램을 실행하면, 링커 또는 로더라고 불리우는 소프트웨어가 프로그램을 하드 디스크로부터 메모리로 불러들인 후 프로그램을 실행하게 됩니다.

# 파이썬 특징

반면에 파이썬은 이러한 컴파일 과정을 필요로 하지 않습니다. 파이썬 프로그램은 파이썬으로된 소스 코드로부터 곧바로 실행됩니다. 이때 파이썬은 내부적으로 실행되는 소스 코드를 '바이트 코드'라고 불리우는 중간 단계의 형태로 변환한 후, 이것을 다시 여러분의 컴퓨터가 사용하는 언어로 변환한 다음 실제로 실행하게 됩니다. 사실 이 모든 과정은, 여러분이 컴파일이라는 과정을 신경쓰지 않고서도, 즉 여러분이 필요한 라이브러리를 갖고 있는지 링크가 잘 되었는지 잘 로드되었는지 등을 신경쓰지 않고서도 파이썬을 쉽게 사용할 수 있게 해 줍니다. 또한 이 특성은 여러분이 작성한 파이썬 프로그램이 여러 플랫폼에서 잘 동작하게 해 주므로, 다른 컴퓨터에 프로그램을

단순히 복사하기만 해도 곧바로 잘 동작하게 됩니다!

# 파이썬 특징

**객체 지향 언어**

- 파이썬은 절차 지향 프로그래밍 및 객체 지향 프로그래밍을 지원
- 절차 지향 언어에서는, 프로그램은 '프로시저' 또는 '함수'들로 구성되는데 이것들은 단순히 프로그램에서 많이 재사용되는 코드 조각들을 의미.
- 반면 객체 지향 언어에서는, 프로그램은 '객체'로 구성되어 있는데 객체란 데이터와 기능이 결합된 하나의 대상을 의미.
- 파이썬은 특히 C++이나 Java와 같은 언어들에 비해 매우 강력하고도 쉬운 방법으로 객체 지향을 지원

# 파이썬 특징

**확장성**

만약 여러분이 작성해야 하는 프로그램의 일부분이 빠른 속도로 동작해야 하거나 혹은 알고리즘의 일부를 공개하고 싶지 않은 경우, 코드의 일부분을 C 혹은 C++로 작성한 후 파이썬 프로그램에서 읽어들여 사용하도록 할 수 있다.

**포함성**

여러분의 C/C++ 프로그램에 파이썬을 포함하도록 하여 여러분의 프로그램을 사용하는 사용자들이 '스크립팅' 기능을 사용하도록 할 수 있다.

# 파이썬 특징

**확장 가능한 라이브러리**

- 파이썬은 방대한 표준 라이브러리를 갖추고 있다.
- 여기에는 정규 표현식, 자동 문서 생성, 유닛 테스트, 쓰레딩, 데이터베이스, 웹 브라우저, CGI, FTP, 전자메일, XML, XML-RPC, HTML, WAV 파일, 암호화 알고리즘, GUI (graphical user interfaces) 등등이 들어 있으며, 여러 시스템 관련 기능들 또한 포함되어 있다.
- 이러한 기능들은 파이썬이 설치되어 있는 어떤 시스템에서든지 사용 가능하다
- 이러한 표준 라이브러리 외에도, 파이썬 패키지 인덱스(Python Package Index)2 에 다양한 라이브러리가 공개

# 파이썬으로 할 수 있는 일

시스템 유틸리티

GUI(Graphic User Interface) 프로그램

C/C++과의 결합

웹 프로그래밍

수치연산 프로그래밍

데이터베이스 프로그래밍

# 딕셔너리

리스트와 비슷하지만 좀 더 일반적입니다. 리스트에서, 지수는 정수만 가능합니다; 딕셔너리에서는 (거의) 모든 형이 가능합니다.

딕셔너리를 지수 (키(key)라고 불립니다) 와 값 간의 사상으로 볼 수 있습니다. 각 키는 값과 대응합니다. 키와 값의 결합을 키-값 쌍(key-value pair) 이나 때로 항목(item)이라고 부릅니다.

# 딕셔너리

다음은 기본적인 딕셔너리의 모습이다.

**{Key1:Value1, Key2:Value2, Key3:Value3 ...}**

즉, Key와 Value쌍들이 여러개가 '{'과 '}'으로 둘러싸이고 각각의 요소는 Key : Value 형태로 이루어져 있고 쉼표(',')로 구분되어져 있음을 볼 수 이다.

다음의 딕셔너리 예를 보도록 하자.

**>>> dic = {'name':'홍길동', 'phone':'0119993323', 'birth': '1118'}**

위에서 key는 각각 'name', 'phone', 'birth'이고 그에 해당하는 value는 '홍길동', '0119993323', '1118'이 된다.

# 절차지향(procedural) 프로그래밍

## 절차지향 프로그래밍

- C언어

물이 위에서 아래로 흐르는 것처럼 순차적인 처리가 중요시 되며
프로그램 전체가 유기적으로 연결되도록 만드는 프로그래밍 기법

## 절차지향의 단점

- 재사용할 수 없다.
- 확장성이 떨어진다.
- 유지보수가 어렵다.

# 객체지향(Object Oriented) 프로그래밍

**프로그래밍 방식의 변화**
초기 : 절차지향 방식
단점 : 조금만 복잡해져도 순서도로 나타내는 것이 불가능한 "스파게티 코드“

1968년 GOTO의 해로움이라는 논문에서 함수(프로시져)단위 호출방식의
구조적 프로그래밍 방식의 등장.
단점 : - 재사용할 수 없다.

이를 극복하기 위한 대안 : **객체 지향 프로그래밍**

# 객체지향(Object Oriented) 프로그래밍

## 객체지향 프로그래밍

- C++, JAVA, Python

구조적 프로그래밍과 다르게 큰 문제를 작게 쪼개는 것이 아니라,
먼저 작은 문제들을 해결할 수 있는 객체들을 만든 뒤,
이 객체들을 조합해서 큰 문제를 해결하는 상향식(Bottom-up) 해결을 도입.

이 객체란 것을 일단 한번 독립성/신뢰성이 높게 만들어 놓기만 하면
그 이후엔 그 객체를 수정 없이 재사용할 수 있으므로 개발 기간과 비용이 대폭 줄어들게 된다.

# 클래스와 객체

클래스는 객체의 틀이 되는 추상적인 개념이고, 객체는 클래스에 정의된 요소들의 실체입니다.

**붕어빵틀 = 클래스**
**붕어빵 = 객체**

# 클래스(class) 개념

```
class Sample:
        pass
```

위의 클래스는 아무런 기능도 없는 클래스이다.
껍질 뿐인 클래스도 인스턴스(instance)라는 것을 생성하는 기능을 갖는다.
(클래스에 의해서 생성된 객체를 인스턴스라고 부른다)
인스턴스는 클래스에 의해서 만들어진 객체로 한개의 클래스는 무수히 많은 인스턴스를 만들어 낼 수가 있다.
Sample 클래스의 인스턴스를 만드는 방법

```
a = Sample()
```

Sample()의 결과값을 돌려 받은 a가 인스턴스 마치 함수를 사용해서 그 결과 값을 돌려 받는 모습과 비슷하다.

# 객체와 인스턴스

## 객체와 인스턴스의 차이

클래스에 의해서 만들어진 객체를 인스턴스라고도 한다.

그렇다면 객체와 인스턴스의 차이는 무엇일까? 이렇게 생각 해 보자. cat = Animal() 이렇게 만들어진 cat은 객체이다. 그리고 cat이라는 객체는 Animal의 **인스턴스(instance)**이다. 즉 인스턴스라는 말은 특정 객체(cat)가 어떤 클래스(Animal)의 객체인지를 **관계**위주로 설명할 때 사용된다. 즉, "cat은 인스턴스" 보다는 "cat은 객체"라는 표현이 "cat은 Animal의 객체" 보다는 "cat은 Animal의 인스턴스" 라는 표현이 훨씬 잘 어울린다.

# 내장 함수

파이썬의 내장 함수는 import 하지 않고 즉시 사용 가능한 함수들이다. 내장 함수명은 일종의 키워드로 간주하여야 하며 사용자의 식별자로 사용하는 것은 피하여야 한다

# 내장 함수

## 입출력 관련 함수

| 함수명 | 기능 |
|---|---|
| print(x) | **객체를 문자열로 표시한다.** |
| input([prompt]) | **사용자 입력을 문자열로 반환한다.** |
| help([x]) | x에 대한 도움말을 출력한다. |
| globals() | **전역 변수의 리스트를 반환한다.** |
| locals() **혹은** vars()<br>vars(obj) | **지역 변수의 리스트를 반환한다.**<br>__dict__ 어트리뷰트를 반환한다. (객체의 내부 변수가 저장된 딕셔너리) |
| del(x) **혹은** del x | **객체를 변수 공간에서 삭제한다.** |
| eval(expr) | **값을 구한다.** |
| exec(obj) | **파이썬 명령을 실행시킨다.** |
| open(filename[,mode])) | **파일을** 연다 |

# 내장 함수

## 기본 자료형의 생성과 변환 함수

| 함수명 | 기능 |
|---|---|
| object() | **새로운** object (모든 객체의 base)를 생성한다. |
| bool(obj) | **객체의 진리값을 반환한다**. |
| int(obj) | **문자열 형태의 숫자나 실수를 정수로 변환한다**. |
| float(obj) | **문자열 형태의 숫자나 정수를 실수로 변환한다**. |
| complex(re [, img]) | **문자열이나 주어진 숫자로 복소수를 생성한다**. |

# 내장 함수

## 기본 자료형의 정보를 얻는 함수

| 함수명 | 기능 |
|---|---|
| type(obj) | **객체의 형을 반환한다**. |
| dir(obj) | **객체가 가진 함수와 변수들을 리스트 형태로 반환한다**. |
| repr(obj)<br>ascii(obj) | evla()함수로 다시 객체를 복원할 수 있는 문자열 생성<br>repr()과 유사하나 non-ascii 문자는 escape한다.(?) |
| id(obj) | **객체의 고유번호**(int형)을 반환한다. |
| hash(obj) | **객체의 해시값**(int형)을 반환. (같은 값이면 해시도 같다.) |
| chr(num)<br>ord(str) | ASCII 값을 문자로 반환<br>한 문자의 ASCII 값을 반환 |
| isinstance(obj, className) | **객체가 클래스의 인스턴스인지를 판단한다**. |
| issubclass(class, classinfo) | class가 classinfo 의  서브클래스일때 True 반환 |

# 내장 함수

## 열거형의 정보를 얻는 함수

| 함수명 | 기능 |
|---|---|
| len(seq) | **시퀀스형을 받아서 그 길이를 반환한다**. |
| iter(obj [,sentinel] )<br>next(iterator) | **객체의 이터레이터**(iterator)를 반환한다.<br>이터레이터의 현재 요소를 반환하고 포인터를 하나 넘긴다. |
| enumerate(iterable, start=0) | **이터러블에서** enumerate 형을 반환한다.<br>입력값으로 시퀀스자료형(리스트, 튜플, 문자열)을 입력을 받는다. |
| sorted(iterable[,key][,reverse]) | **정렬된 *리스트*를 반환** |
| reversed(seq) | **역순으로 된** *iterator*를 반환한다. |
| filter(func, iterable) | iterable의 각 요소 중 func()의 반환값이 참인 것만을 묶어서 이터레이터로 반환. |
| map(func, iterable) | iterable의 각 요소를 func()의 반환값으로 매핑해서 이터레이터로 반환. |

**iterable:반복가능한 자료형 (문자열, 리스트, 튜플, 딕셔너리, 집합)**

# 내장 함수

## 산술/논리 연산에 관련된 함수

| | |
|---|---|
| hex(n)<br>oct(n)<br>bin(n) | **정수** n의 16진수 값을 구해서 '문자열'로 반환한다.<br>정수 n의 8진수 값을 구해서 '문자열'로 반환한다.<br>정수 n의 2진수 값을 구해서 '문자열'로 반환한다. |
| abs(n) | **절대값을 구한다**. 복소수의 경우 크기를 구한다. |
| pow(x,y[,z]) | **거듭제곱을 구한다**. pow(x,y)은 x**y 와 같다. |
| divmod(a,b) | a를 b로 나눈 (몫, 나머지)를 구한다. 튜플 반환. |
| all(iterable)<br>any(iterable) | iterable 의 모든 요소가 True 일 경우 True를 반환.<br>iterable 의 하나 이상의 요소가 True 일 경우 True를 반환. |
| max(iterable)<br>max(arg1, arg2, …) | **최대값을 구한다**. |
| min(iterable)<br>min(arg1, arg2, …) | **최소값을 구한다**. |
| round() | **반올림을 한다**. |