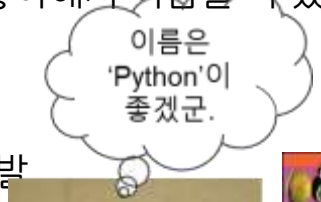
Python 개요

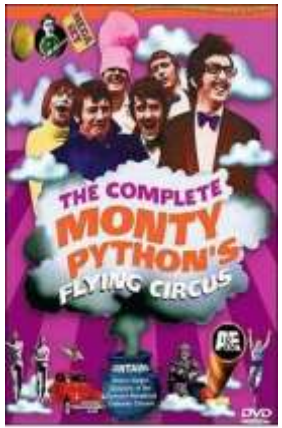
김지성 강사

파이썬 이란?

- ✓ **귀도 반 로섬(Gudi Van Rossum)** 개발
- ✓ 파이썬은 **인터프리터 언어**
- ✓ BBC 방송 코미디 쇼 (Monty **Python's** Flying Circus) 를 좋아해서 이름을 따 왔음
- ✓ 1989년 개발시작 1990년 첫 버전 공개
  - ○ 크리스마스에 "취미"가 될만한 프로그램을 찾다가 개발
- ✓ 대형 글로벌기업부터 스타트업까지 다양하게 활용



파이썬 창시자
귀도 반 로섬



Monty Python

파이썬 이란?

- ✓ **사람이 이해하기에 문법이 쉽다**. → 사람의 언어와 흡사하다.
- ✓ 데이터를 다루기 위한 NumPy, Pandas 등 **제공되는 라이브러리가 많다**.
- ✓ 손쉽게 데이터를 시각화할 수 있는 Matplotlib, Seaborn 라이브러리
- ✓ PyQT, Tkinter 등 GUI로 앱을 개발할 수 있다.
- ✓ Django, Flask 프레임워크를 활용하여 웹 서비스를 만들 수 있다.
- ✓ SciPy를 활용하여 **과학기술 계산 및 알고리즘을 활용**할 수 있다.
- ✓ **Tensorflow, Keras, PyTorch를 활용하여 Deep Learning 모델을 구현**할 수 있다

파이썬 이란?

- ✓ 파이썬의 철학
- ✓ Zen of Python

tip

Beautiful is better than ugly. 아름다움이 추한 것보다 낫다.

Explicit is better than implicit. 명확함이 함축된 것보다 낫다.

Simple is better than complex. 단순함이 복잡한 것보다 낫다.

Complex is better than complicated. 복잡함이 난해한 것보다 낫다.

Flat is better than nested. 단조로움이 중첩된 것보다 낫다.

Sparse is better than dense. 여유로움이 밀집된 것보다 낫다.

Readability counts. 가독성은 중요하다.

Special cases aren't special enough to break the rules. 규칙을 깨야할 정도로 특별한 경우란 없다. Although practicality beats purity. 비록 실용성이 이상을 능가한다 하더라도.

Errors should never pass silently. 오류는 결코 조용히 지나가지 않는다. Unless explicitly silenced. 알고도 침묵하지 않는 한.

In the face of ambiguity, refuse the temptation to guess. 모호함을 마주하고 추측하려는 유혹을 거절하라. There should be one-- and preferably only one --obvious way to do it. 문제를 해결할 하나의 - 바람직하고 유일한 - 명백한 방법이 있을 것이다. Although that way may not be obvious at first unless you're Dutch. 비록 당신이 우둔해서 처음에는 명백해 보이지 않을 수도 있겠지만.

Now is better than never. 지금 하는 것이 전혀 안하는 것보다 낫다. Although never is often better than *right* now. 비록 하지않는 것이 지금 하는 것보다 나을 때도 있지만.

If the implementation is hard to explain, it's a bad idea. 설명하기 어려운 구현이라면 좋은 아이디어가 아니다. If the implementation is easy to explain, it may be a good idea. 쉽게 설명할 수 있는 구현이라면 좋은 아이디어일 수 있다. Namespaces are one honking great idea -- let's do more of those! 네임스페이스는 정말 대단한 아이디어다. -- 자주 사용하자!

파이썬 이란?

- ✓ C++ vs Java vs Python
- ✓ 코드의 간결함 비교

C++

```
#include <iostream>
using namespace std;
int main() {
   cout<<"Hello, gabia!";
   return 0;
}
```

Java

```
public class HelloGabia {
   public static void main(String args[]) {
     System.out.println("Hello, gabia!");
   }
}
```

Python

```
print("Hello, gabia!")
```

파이썬 이란?

✓ 파이썬으로 개발하는 기업 CWN 뉴스

파이썬의 특징

- ✓ 동적 타이핑(Dynamic typing)을 지원
- ✓ 동적 타이핑은 **데이터의 타입을 명시하지 않아도 컴퓨터가 스스로 분석**해서 타입에 대한 처리를 하도록 하는 방식
  - ○ 매번 데이터마다 어떤 타입인지 결정하기 위한 프로세스 때문에 속도는 느릴 수 있으나, 코드를 작성할 때는 편리
- ✓ 이와는 반대로 정적 타이핑(Static typing)의 대표적인 언어는 C, C++,JAVA

Java

```java
public class HelloTypes {

    public static void main(String[] args) {

        String thing;
        thing = "Hello World";

        System.out.println(thing);
    }
}
```

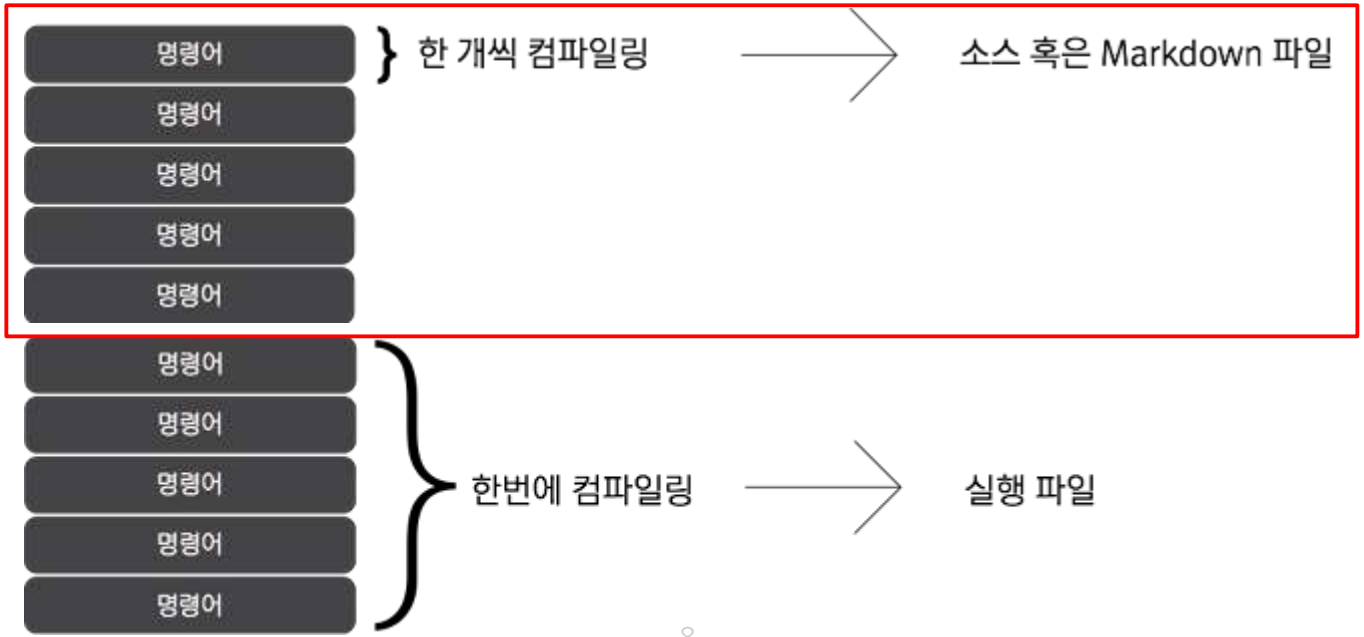
Python

```
>>> thing = "Hello"
>>> type(thing)
<class 'str'>

>>> thing = 28.1
>>> type(thing)
<class 'float'>
```

파이썬의 특징

- ✓ **인터프리터(Interpreter) 방식으로 동작**하는 언어
- ✓ 프로그래밍 언어로 된 코드를 컴퓨터가 바로 처리할 수 있도록 즉시 변환
- ✓ 다른 방식으로 컴파일 방식이 존재



파이썬의 특징

- ✓ 파이썬 소스 파일의 **기본 인코딩은 UTF-8**
- ✓ 인코딩이란 컴퓨터에서 문자를 처리하기 위한 방식으로 각 문자마다 0/1로 매핑이 되어있음
- ✓ 이러한 약속은 표준이 여럿일 수 있는데, 그 중 하나가 ASCII(아스키)이고 다른 하나는 UTF-8
- ✓ ASCII는 영어권에서 컴퓨터가 만들어져서 그 때 사용되던 표준으로 2^7=128개를 가지고 있음
- ✓ UTF-8은 다양한 언어의 표현을 위해 만든 표준

UTF-8 encoding table and Unicode characters

U+003F	?	3f	QUESTION MARK
U+0040	@	40	COMMERCIAL AT
U+0041	A	41	LATIN CAPITAL LETTER A
U+0042	B	42	LATIN CAPITAL LETTER B
U+0043	C	43	LATIN CAPITAL LETTER C
U+0044	D	44	LATIN CAPITAL LETTER D
U+0045	E	45	LATIN CAPITAL LETTER E
U+0046	F	46	LATIN CAPITAL LETTER F
U+0047	G	47	LATIN CAPITAL LETTER G
U+0048	H	48	LATIN CAPITAL LETTER H
U+0049	I	49	LATIN CAPITAL LETTER I
U+004A	J	4a	LATIN CAPITAL LETTER J
U+004B	K	4b	LATIN CAPITAL LETTER K
U+004C	L	4c	LATIN CAPITAL LETTER L
U+004D	M	4d	LATIN CAPITAL LETTER M
U+004E	N	4e	LATIN CAPITAL LETTER N

파이썬을 들어가기 전 지켜야 할 사항

- ✓ **세미콜론은 옵션**입니다.
- ✓ 파이썬의 경우는 생략이 가능. **한 줄에 여러 명령을 작성할 경우에만** 구분자로 사용하면 됨.
  - ○ java: System.out.println(“hello world”);
  - ○ python: print(“hello”)
- ✓ 파이썬은 **띄어쓰기(공백)**를 통해 **코드의 범위를 구분**합니다.
  - ○ 들여쓰기를 통해 구분한다는 뜻!
- ✓ 즉, **띄어쓰기를 할 땐 늘 주의**해서 구획(코드 블록)이 섞이지 않도록 해야함.
- ✓ 범위를 나눌 때 **띄어쓰기 4회를 권장**하고 있음.(PEP 8) 

포맷팅

김지성 강사

포맷팅

- ✓ 파이썬은 **복잡한 문자열 출력을 위한 문자열 형식화(string formatting)를 지원**한다.
- ✓ 문자열을 형식화하는 방법은 다음과 같다.
  - ○ 서식 지정자인 **%** 기호를 사용하는 방식
  - ○ **format 메소드**를 사용하는 방식
  - ○ **f-string**을 사용하는 방식

포맷팅 - 서식 지정자

✓ % 기호를 사용하는 방식

형식지정 문자열	의미
%s	문자열
%d	정수
%f	부동소수점 실수

포맷팅 - 서식 지정자

✓ % 기호를 사용하는 방식

```
print("내 이름은 %s입니다." % "홍길동")

print("나는 %d살 입니다." % 12)

print("원주율의 값은 %f입니다." % 3.141592)

print("원주율의 값은 %.2f입니다." % 3.141592)

print("%d 곱하기 %d은 %d이다." % (2, 3, 6))

print("%s의 %s 과목 점수는 %d점이다." % ("홍길동", "수학", 100))
```

포맷팅 - 서식 지정자

✓ 연습 문제1

✓ 동일한 출력 결과를 보이게끔 만들어보세요.

```
print("이름은 ?, 나이는 ?, 키는 ? 입니다." ?)

# TODO: ?를 올바르게 채워넣으세요.
```

이름은 홍길동, 나이는 20, 키는 171.1 입니다.

포맷팅 - format 메소드

✓ format 메소드를 사용하는 방식

- % 기호 대신 {} 기호를 사용
- 자료형을 표시할 필요가 없음

```
print("내 이름은 {}입니다.".format("홍길동"))

print("내 이름은 {{}}입니다.".format("홍길동"))

print("{2}의 {0} 점수는 {1}점입니다. {1}점! {1}점!".format("수학", 100, "철수"))

print("{a}점수: {x}점, {b}점수: {y}점".format(a="영어", b="수학", x=100, y=90))

print("{}점수: {}점, {}점수: {}점".format("영어", 100, "수학", 90))

print("{:.3f}".format(1/3))
```

포맷팅 - format 메소드

- ✓ 연습 문제1
- ✓ 동일한 출력 결과를 보이게끔 만들어보세요.

```
print("이름은 ?, 나이는 ?, 키는 ?입니다." ? )

# TODO: ?를 올바르게 채워넣으세요.
```

이름은 홍길동, 나이는 20, 키는 171.1 입니다.

포맷팅 - f-string

✓ f-string을 사용하는 방식

- ○ 파이썬 3.6부터는 f-string을 사용할 수 있다.
- ○ f-string은 문자열의 앞에 f 글자를 붙이고 {} 안에 변수의 이름을 바로 사용 가능.

```
math = "수학"
student = 20
score = 84.51

print(f"{math} 수업을 듣는 학생은 총 {student}명이고, 나의 점수는 {score}다. 반올림을 하면 {score:.0f}이다.")
```

포맷팅 - f-string

✓ 연습 문제1

✓ 동일한 출력 결과를 보이게끔 만들어보세요.

```
name = "홍길동"
age = 20
height = 171.12

print(f"이름은 ?, 나이는 ?, 키는 ?입니다.")


# TODO: ?를 올바르게 채워넣으세요.
```

이름은 홍길동, 나이는 20, 키는 171.1 입니다.

키워드, 식별자, 변수, 상수, 리터럴

김지성 강사

키워드(Keyword)

✓ 키워드(Keyword)는 **파이썬에서 이미 예약되어있는 문자열**이다.

✓ 따라서 다른 용도로 사용이 불가능하다.

```
import keyword
display(keyword.kwlist)
```

✓ 출력결과를 보면 False, True … 등과같은 문자열은 파이썬 고유의 키워드를 볼 수 있다.

식별자(Identifier)

- ✓ 식별자(Identifier)는 변수, 상수, 함수, 사용자 정의 타입등에서 다른 것들과 구분하기 위해서 사용되는 **변수의 이름, 상수의 이름등을 일반화 해서 지칭**하는 용어.
- ✓ 사용자가 임의로 지정하지만 **직관적이고 가독성있게 정의**하는 것이 좋다.
- ✓ 식별자 명칭의 작성 스타일로는 다음과 같은 3가지가 존재. (헝가리안 표기법은 사용하지 않음)
  - ○ 카멜 표기법(Camel Case) : firstName, lastName
  - ○ 파스칼 표기법(Pascal Case) : FirstName, LastName
  - ○ 스네이크 표기법(Snake Case) : first_name, last_name

식별자(Identifier)

✓ 대부분 카멜 표기법과 파스칼 표기법을 적절하게 조합하여 사용.

✓ Python에서는 “Style Guid for Python Code”를 기술하고 있음

- 코드 배치, 표현식과 구문의 공백등 다양한 코드 스타일을 기술하고 있음.
- 파이썬 개발자들은 이러한 규칙을 가지고 코드를 작성하기 때문에 앞으로 우리도 이러한 규칙들을 지켜서 코드를 작성하면 좋다.

식별자(Identifier)

✓ 주요 파이썬 코드의 특징

- **함수와 변수는 snake_case**를 사용한다. (소문자로 시작)
- **파이썬에서는 camel case를 사용하지 않는다. 클래스를 작성할때는 pascal case로 작성**.
  - PEP8에서는 upper camel case라고 표현한다.
- **1개의 밑줄로 시작** : 클래스 내부적으로 사용되는 속성 및 메소드
  - 외부 접근은 가능하지만 내부적으로만 사용하기로 암묵적인 약속
- **1개의 밑줄로 종료** : 파이썬 기본 키워드와의 충돌을 피하기 위해 사용
- **2개의 밑줄로 시작** : 클래스 내부의 속성 이름 충돌을 방지
  - 속성과 메소드를 보호하고 이름 충돌을 방지 (Name Mangling)
- **2개의 밑줄로 시작 및 종료** : python 내장 메소드와 속성을 나타낼 때 사용 (특수 메소드)

식별자(Identifier)

✓ 1개의 밑줄로 시작하는 경우

```
class MyClass:
    def __init__(self):
        self._internal_var = 42  # 내부에서만 사용하는 변수

    def _internal_method(self):  # 내부적으로 사용하는 메서드
        print("This is an internal method.")

# 사용 예시
obj = MyClass()
# 외부에서 접근할 수는 있지만 내부적으로만 사용하길 권장
print(obj._internal_var)  # 42
obj._internal_method()  # "This is an internal method."
```

식별자(Identifier)

✓ 1개의 밑줄로 종료하는 경우

```
class MyClass:
    def __init__(self):
        self.class_ = "reserved keyword"  # 'class'와 충돌 방지를 위해 '_'

# 사용 예시
obj = MyClass()
print(obj.class_)  # "reserved keyword"
```

식별자(Identifier)

✓ 2개의 밑줄로 시작하는 경우

```
class MyClass:
    def __init__(self):
        self.__private_var = 42  # Name mangling 처리됨

    def __private_method(self):  # Name mangling 처리됨
        print("This is a private method.")

# 사용 예시
obj = MyClass()
# 외부에서 접근 불가능
# print(obj.__private_var)  # AttributeError 발생
# obj.__private_method()  # AttributeError 발생

# Name mangling으로 접근 가능 (잘 사용하지 않음)
print(obj._MyClass__private_var)  # 42
obj._MyClass__private_method()  # "This is a private method."
```

식별자(Identifier)

✓ 2개의 밑줄로 종료하는 경우

- ○ __init__의 초기화 메소드를 사용자가 정의
- ○ __lein__의 반환값을 사용자가 정의

```
class MyClass:
    def __init__(self, value):
        self.value = value  # 초기화 메서드

# 사용 예시
obj = MyClass(10)
print(obj.value)  # 10
class MyList:
    def __init__(self, items):
        self.items = items

    def __len__(self):  # len() 호출 시 반환값 정의
        return len(self.items)

# 사용 예시
my_list = MyList([1, 2, 3])
print(len(my_list))  # 3
```

변수(Variable)

- ✓ 컴퓨터는 보조기억 장치(SSD,HDD…) -> 주기억 장치(RAM) -> 중앙처리장치(CPU)의 순서로 데이터를 읽고 처리함.
- ✓ **주기억 장치(RAM)은 Byte마다 번지가 지정**되어 있는데, 이를 **메모리 주소**라고 말함.
  - ○ 10번지에 있는 데이터를 읽어서 20번지에 있는 데이터와 더해서 30번지에 대입해
- ✓ 메모리 관리는 운영체제가 맡아서 하고 프로그래머가 변수를 선언하고 값을 할당하면 됨
  - ○ 운영체제가 현재 사용하지 않는 메모리를 확보해서 그곳에 변수명을 할당
- ✓ 즉 **변수는 어떤 값을 담는 상자**라고 볼 수 있다.
- ✓ 대입연산자를 통해 변수에 값을 할당할 수 있음.

```
var1 = 1
var2 = "test"
print(f"var1 의 값 : {var1}, var2의 값 : {var2}")
```

변수(Variable)

- ✓ 사진과 같이 메모리 주소가 있다고 가정한다면 각 메모리 주소에는 특정 객체를 저장하고 있다.
- ✓ 무작위로 선택된 메모리 위치에 객체를 저장하고, 파이썬은 이 메모리 위치에 접근하여 값을 찾는다.
- ✓ python의 id() 메소드는 객체가 저장된 주소를 반환

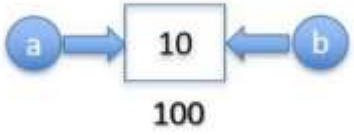
변수(Variable)

✓ 파이썬 vs C/C++ 변수 메모리의 비교

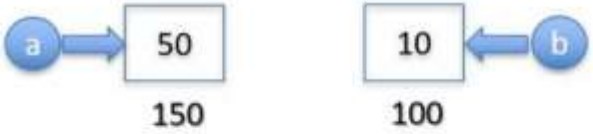
변수의 개념은 Python에서 C/C++와 다르게 작동합니다. C/C++에서 변수는 명명된 메모리 위치입니다. a=10이고 b=10이면 둘 다 서로 다른 두 메모리 위치입니다. 메모리 주소가 각각 100과 200이라고 가정해 보겠습니다.

"a"에 다른 값(예: 50, 10)을 할당하면 주소 100에 있는 값이 덮어쓰여집니다.

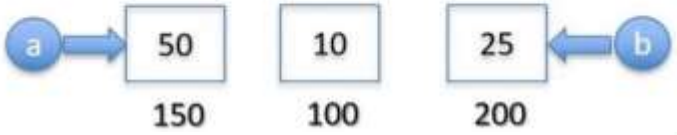
파이썬 변수는 메모리 위치가 아닌 객체를 참조합니다. 객체는 메모리에 한 번만 저장됩니다. 여러 변수는 실제로 동일한 객체에 대한 여러 레이블입니다.



명령문 a=50은 다른 위치의 메모리에 새로운 **int 객체 50을 생성하고, "b"가 참조하는 객체 10은 그대로 둡니다.**



또한, b에 다른 값을 할당하면 객체 10은 참조되지 않은 상태로 유지됩니다.



변수(Variable)

✓ 파이썬 vs C/C++ 변수 메모리의 비교

```
a=10
b=10
print(id(a) == id(b))

b=50
print(id(a) == id(b))
```

```
True
False
```

상수(Constant)

✓ 상수(constant)는 항상 똑같은 값을 저장한다. 즉 사전에 미리 정의가 되어있음.

✓ constant1.py -> 파일 이름

```
PI = 3.14
GRAVITY = 9.8
```

✓ python_basic1.ipynb -> 파일 이름

```
import constant1

print(constant1.PI)
print(constant1.GRAVITY)
constant1.PI = 3.141592 # 값을 변경하므로 상수가 역할이 사라짐
print(constant1.PI)
```

상수(Constant)

✓ constant2.py -> 파일 이름

```
class _constant2:
    def __setattr__(self, name, value):
        if name in self.__dict__:
            raise Exception('변수에 값을 할당할 수 없습니다.')
        self.__dict__[name] = value

    def __delattr__(self, name):
        if name in self.__dict__:
            raise Exception('변수를 삭제할 수 없습니다.')

import sys
sys.modules[__name__] = _constant2()
```

상수(Constant)

✓ python_basic1.ipynb -> 파일 이름

```
import constant2

constant2.PI = 3.14
constant2.MAX = 1000
print(constant2.PI)
print(constant2.MAX)

# 재할당시 에러발생
constant2.PI = 3.141592
constant2.MAX = 2000
print(constant2.PI)
print(constant2.MAX)
```

✓ 에러를 발생시키는데 실제로 파이썬에서 상수를 만들 때 위와 같은 방법으로 만들 수 있다.

✓ 클래스에 대한 내용은 뒤에서 자세히 배우고 지금은 상수의 개념에 대해서만 알아두도록 하자.

리터럴(Literal)

✓ 리터럴(Literal)은 “값” 그 자체를 의미한다.

✓ 파이썬은 숫자, 문자, 논리, 특수, 컬렉션 리터럴이 존재한다.

```
# 숫자 리터럴
integer_literal = 100 # Decimal Literal
print("정수 리터럴", integer_literal)

# 문자 리터럴
strings_literal = "This is Python"
print("문자 리터럴", strings_literal)

# 논리 리터럴
bool_literal = (1 == True)
print("논리 리터럴", bool_literal)

# 특수 리터럴
special_literal = None
print("특수 리터럴", special_literal)
```

리터럴(Literal)

- ✓ 문자열 리터럴(Literal)은 다양하게 활용이 가능하다.
- ✓ 줄바꿈을 진행하려면 \n을 사용하면 됨

```
s = 'First line. \nSecond line.'
print(s)
```

```
First line.
Second line.
```

리터럴(Literal)

- ✓ 문자열 리터럴(Literal)은 다양하게 활용이 가능하다.
- ✓ \n을 그대로 출력하려면
  - ○ raw string으로 표현하기. r을 맨 앞에 붙이면 된다.
  - ○ 이스케이프 문자\를 앞에 붙여 문자를 그대로 출력.

```
s1 = r'c:\name'
s2 = 'c:\\name'
print(s1)
print(s2)
```

```
c:\name
c:\name
```

리터럴(Literal)

✓ 문자열 리터럴(Literal)은 다양하게 활용이 가능하다.

✓ 삼중 따옴표를 사용하면 여러 줄로 확장하여 문자열 리터럴을 만들 수 있음.

- ○ **삼중 따옴표내에 존재하는 텍스트 전체를 그대로 표현**함.
- ○ 개행을 표시하고 싶지 않다면 \를 사용해서 방지할 수 있음.

```
s3 = """
안녕하세요.
오늘은 파이썬 수업을 하고 있습니다.
\n, \\,
"""
print(s3)
```

```
안녕하세요.
오늘은 파이썬 수업을 하고 있습니다.
\n, \\,
```

연습 문제

✓ 다음과 같은 출력문이 나오게끔 출력해보세요.

```
Hello, world!
Hello, world!
Python Programming
```

✓ var1, var2 변수에 적절한 값을 넣어 아래와 같은 print문을 입력했을 때 동일한 출력문이 나오게 만들어보세요.

```
print(var1,var2)
```

```
hello world!
```

연습 문제

✓ 다음과 같은 출력문이 나오게끔 출력해보세요. (삼중 따옴표를 활용)

```
안녕하세요.
오늘은 파이썬 수업을 하고 있습니다.
즐거운 시간이 될 것 같아요!
```

✓ 다음과 같은 출력문이 나오게끔 출력해보세요. (삼중 따옴표를 활용)

```
"안녕하세요".
'오늘'은 파이썬 수업을 하고 있습니다.
즐거운 시간이 될 것 같아요!
```