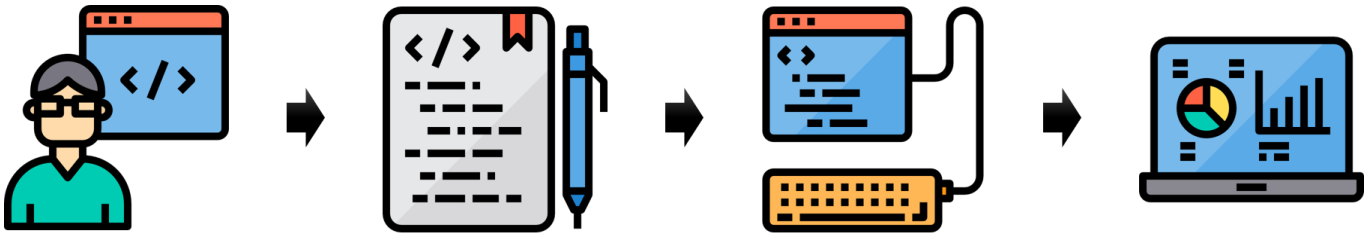
파이썬 프로그래밍 언어(Python Programming Language)

프로그래밍 언어 개념



- 프로그래머(programmer): 프로그래밍 언어를 사용해 소프트웨어를 만드는 사람
- 프로그래밍 언어(programming language): 컴퓨터 시스템을 구동시키는 소프트웨어를 작성하기 위한 형식언어
- 프로그래밍(programming): 프로그래밍 언어를 사용하여 프로그램을 개발하는 것

파이썬 소개(Python Introduction)

- 귀도 반 로섬(Guido Van Rossum)이 제작해 1991년에 공식 발표한 고급 프로그래밍 언어

- 플랫폼 독립적이며 인터프리터식, 객체지향적, 동적 타이핑(dynamic typed) 대화형 언어
- 파이썬이라는 이름은 귀도가 좋아하는 코디미 <Monty Python's Flying Circus>에서 따온 것
- 파이썬의 사전적인 의미는 비단뱀으로 로고 디자인도 파랑색과 노란색 비단뱀 두 마리가 서로 얽혀 있는 형태

- 파이썬 프로그래밍 언어는 Google Trends를 통해 분석한 결과, 2020년 기준 가장 인기있는 언어 1위 (PYPL:PopularitY of Programming Language: http://pypl.github.io/PYPL.html)

파이썬 특징(Python Features)

쉬운 문법과 간결함

- 사람의 사고 체계와 유사하고 문법 자체가 아주 쉽고 간결함
- 소스 코드가 이해하기 쉽어 공동 작업과 유지 보수가 쉬움

동적 타이핑(dynamic typing) 범용 프로그래밍 언어

- 프로그램의 실행 시점에서 프로그램 변수의 타입을 결정하는 언어

다양한 플랫폼 지원과 호환성

- 다양한 플랫폼에서 동작하도록 지원
- C언어와의 호환성이 매우 뛰어남

풍부한 라이브러리(모듈)

- 다양한 파이썬 내부 라이브러리 뿐만 아니라 다양한 서드 파티third party 라이브러리 사용 가능

풀 언어(glue language)

- 다른 언어로 쓰인 모듈들을 연결하는 언어로 자주 이용
- C나 C++ 언어로 만든 프로그램을 파이썬에서 사용 가능

다양한 언어의 문자 처리

- 유니코드 문자열을 지원해서 다양한 언어의 문자 처리 가능

파이썬 응용(Python Applications)

시스템 유틸리티

- 인증 / 권한 관리
- 캐시 관리 / 파일 시스템 도구
- 코드 분석 / 빌드, 디버깅 도구

GUI(Graphic User Interface) 개발

- 윈도우 형태의 프로그래밍 가능
- Tkinter 패키지

웹 프로그래밍

- 웹에서 회원 가입, 게시판, 블로그 등의 프로그램 제작 가능
- Django, Flask 등 파이썬 기반 웹 프레임워크

데이터베이스

- Oracle, MySQL, PostgreSQL 등의 데이터베이스에 접근 가능한 도구 제공
- 데이터베이스를 이용한 다양한 프로그램 제작 가능

데이터 분석 및 시각화

- 데이터 처리, 탐색, 분석, 시각화까지 구현 가능
- NumPy, Pandas, DataFrame, Matplotlib 패키지

기계학습/딥러닝 모델

- 모델 생성과 데이터 학습 등을 위한 편리한 도구 제공
- Scikit-learn, TensorFlow, PyTorch, Keras 패키지

이미지 처리/컴퓨터 비전

- 컴퓨터 비전과 관련된 다양한 처리와 알고리즘을 제공
- OpenCV 패키지

자연어 처리

- 텍스트 데이터나 자연어 처리를 위한 다양한 모델 제공
- WordCloud, Gensim 패키지

게임 개발

- 게임 개발에 필요한 이미지, 소리, 입력 장치 등의 효율적인 처리 방법을 제공
- PyGame 패키지

기타

- 로봇, 사물인터넷, 클라우드 컴퓨팅 등 수 많은 분야에서 다양한 응용으로 활용
- 자세한 내용들은 Awesome Python(https://awesome-python.com) 참고

파이썬 설치 및 환경

- 파이썬을 이용하는 방법은 다양하게 존재하며 사용하기 원하는 환경에 따라서 선택
- 개인 컴퓨터에서 파이썬 사용시에는 홈페이지(https://www.python.org) 에서 설치 파일을 다운로드하여 설치
- 파이썬 기본 에디터보다는 다양한 통합개발환경(IDE)을 활용하여 프로그래밍 가능
- 대표적인 통합개발환경(IDE): PyCharm(https://www.jetbrains.com/pycharm), Visual Studio Code(https://code.visualstudio.com)
- 본 강의에서는 구글 클라우드 환경인 Google Colab(https://colab.research.google.com) 에서 파이썬을 쉽게 배우고 실습

파이썬 시작하기

- 프로그래밍 언어를 배울 때 보통 가장 먼저 하는 것은 인사
- 가장 기본이 되는 화면 출력 함수인 `print()`를 이용해서 문자열 "Hello Python"을 출력

```
print("Hi Python")
```

```
Hi Python
```

파이썬으로 계산하기

사칙연산

- 파이썬에서는 기본적인 산술 연산을 바로 수행 가능
- 더하기, 빼기, 곱하기, 나누기 같은 간단한 산술연산 예제

```
1 + 1
```

```
2
```

```
5 - 1
```

```
4
```

```
4 * 3
```

```
12
```

```
3 / 5
```

```
0.6
```

실수 사칙연산

- 실수에 대해서도 사칙연산 가능

```
1.1 + 2.4
```

```
3.5
```

```
4.2 - 1.2
```

```
3.0
```

```
1.2 * 2.2
```

```
2.64
```

```
5.4 / 2.7
```

```
2.0
```

복잡한 연산

- 여러 산술연산들이 포함된 복잡한 연산들도 가능

```
1 + 2 + 3 / 3 * 2
```

```
5.0
```

```
4 / 2 * 2 - 10 / 4
```

```
1.5
```

```
(10 - 2 * 4) / (1 + 2) * (10 - 4)
```

```
4.0
```

파이썬 문법(Python Syntax)

들여쓰기(Indentation)

- 코드 라인의 시작 부분에 들여쓰기
- 파이썬은 들여쓰기로 코드의 블록을 표시

```
if 2 > 1:
  print(2)
```

```
    2
```

```
if 2 > 1:
  print(2)
```

```
    2
```

주석(Comments)

- 코드에 대해 설명하는데 사용하고 가독성 증가
- 일부 코드를 주석처리하여 실행 방지 가능

```
# 주석 부분
print("주석문") # 출력
#print("Comment")
```

```
    주석문
```

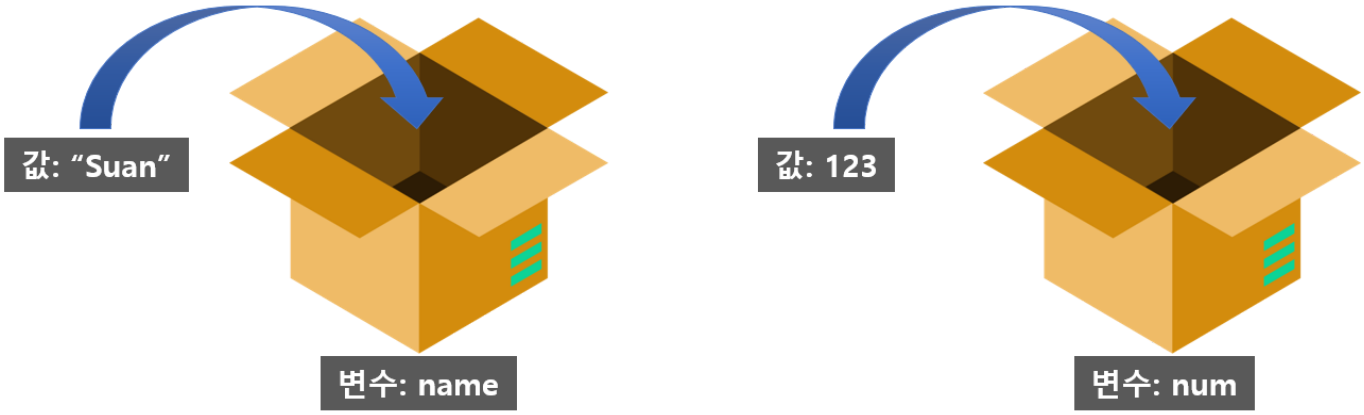
변수, 자료형, 연산자

변수(Variables)

변수 개념

- 어떠한 값을 저장하는 공간(메모리)
- 일반적으로 데이터의 저장 위치와 데이터 값으로 구분
- 변수에 값을 넣는 순간 메모리 저장 위치를 할당하고 그 위치는 메모리 주소로 관리

실제 변수에 대해서 실습하기 위해 그림과 같이 변수 `name`에 문자열 `"Suan"`를 넣고, 변수 `num`에 값 `123`을 넣기




```
name = "Suan"
print(name)
num = 123
print(num)
```

```
Suan
123
```

변수명 규칙

- 알파벳, 숫자, 언더바(_)로 선언 가능
- 의미 있는 단어로 표기하는 것이 좋음
- 대소문자가 구분
- 특별한 의미가 있는 예약어는 사용할 수 없음

예약어

예약어					
and	exec	not	assert	finally	or
break	for	pass	class	from	print
continue	global	raise	def	if	return
del	import	try	elif	in	while
else	is	with	except	lambda	yield

```
abc = "abc"
abc_123 = 123
Abc_123 = 123
print(abc)
print(abc_123)
print(Abc_123)
```

```
abc
123
123
```

자료형(Data Types)

- 파이썬에서 자료형은 유형에 따라서 논리형, 수치형, 문자형, 구조형으로 구분
- 일반적으로 숫자와 문자, 그리고 구조에 따른 저장을 위해서 여러가지 구분된 자료형을 제공

유형	자료형	설명	선언
논리형	불리언(boolean type)	참(True) 또는 거짓(False)을 표현할 때 사용	`b = True`
수치형	정수(integer type)	자연수를 포함해 값의 영역이 정수로 한정된 값	`i = 10`
수치형	2진수 정수(binary type)	2진수 자료형(숫자가 '0b'또는 '0B'로 시작)	`b = 0b010`
수치형	8진수 정수(octal type)	8진수 자료형(숫자가 `0o` 또는 `00` 로 시작)	`o = 0o130`
수치형	16진수 정수(hexadecimal type)	16진수 자료형(`0x` 로 시작)	`h = 0xABC`
수치형	실수(floating-point type)	소수점이 포함된 값	`f = 12.34`
수치형	복소수(complex type)	복소수 사용을 위한 자료형	`j = 1 + 23j`
문자형	문자열(string type)	값이 문자로 출력되는 자료형	`s = "Suan"`
시퀀스형	리스트(list type)	여러 요소를 묶어 하나의 변수로 사용	`l = [1, 2, 3]`
시퀀스형	튜플(tuple type)	리스트와 유사하지만 생성, 삭제, 수정 불가	`t = (1, 2, 3)`
시퀀스형	범위(range type)	숫자의 불변한 시퀀스 형태	`range(10)`
집합형	집합(set type)	중복과 순서가 없는 집합을 위한 자료형	`s = {1, 2, 3}`
집합형	불변 집합(fronzenset type)	불변한 집합을 위한 자료형	`fs = frozenset{1, 2, 3}`
매핑형	딕셔너리(dictionary type)	키(key)와 값(value)이 쌍(pair)으로 들어간 자료형	`d = {1:'One', 2:'Two'}`
바이트 시퀀스형	바이트(byte type)	바이트 값을 가지는 자료형	`byte = b'hello'`
바이트 시퀀스형	bytearray type	바이트의 시퀀스 형태로 수정 가능한 자료형	`ba = bytearray(b'hello')`
바이트 시퀀스형	memoryview type	바이트의 메모리 값으로 표현한 자료형	`mv = memoryview(b'hello'`

불리언형 자료형(Boolean Type)

- 불리언은 `True`와 `False` 값으로만 표현할 때 사용하는 자료형
- 예제 코드에서 변수 `b`에 `True` 값을 넣으면 불리언 자료형으로 결정
- 자료형을 반환하는 내장함수 `type()` 함수를 이용하여 자료형을 확인하면 `bool`로 반환된 결과를 볼 수 있음

```
b = True
print(b)
print(type(b))
```

```
True
<class 'bool'>
```

정수형 자료형(Integer Type)

10진수(Decimal)

- 10진수는 일반적인 숫자 표현을 그대로 사용
- 변수 `i`에 10진수로 숫자 10을 값으로 넣고, `type()` 함수를 통해 자료형을 확인하기
- 결과는 당연히 정수형(int) 자료형으로 출력

```
i = 10
print(i)
print(type(i))
```

```
10
<class 'int'>
```

2진수(Binary)

- 2진수는 앞에 `0b`를 붙여서 표현
- 변수 `b`에 2진수 표현으로 `0b010`을 넣고, `type()` 함수를 통해 자료형을 확인하기
- 2진수 010이 10진수로 변환되는 계산: $010_{(2)} = 0 \times 2^2 + 1 \times 2^1 + 0 \times 2^0 = 2_{(10)}$

```
b = 0b010
print(b)
print(type(b))
```

```
2
<class 'int'>
```

8진수(Octal)

- 8진수는 앞에 `0o`를 붙여서 표현
- 변수 `o`에 8진수 표현으로 `0o130`을 넣고 확인하기
- 8진수 130이 10진수로 변환되는 계산: $130_{(8)} = 1 \times 8^2 + 3 \times 8^1 + 0 \times 8^0 = 88_{(10)}$

```
o = 0o130
print(o)
print(type(o))
```

```
88
<class 'int'>
```

16진수(Hexadecimal)

- 16진수은 앞에 `0x`를 붙여서 표현
- 변수 `h`에 16진수 표현으로 `0xABC`를 넣고 확인하기
- 16진수는 10진수와 달리 10부터 A에서 F까지 알파벳을 사용하여 표현:
  $A_{(16)} = 10_{(10)}, B_{(16)} = 11_{(10)}, C_{(16)} = 12_{(10)}, D_{(16)} = 13_{(10)}, E_{(16)} = 14_{(10)}, F_{(16)}$

- 16진수 ABC가 10진수로 변환되는 계산:
  $ABC_{(16)} = A \times 16^2 + B \times 16^1 + C \times 16^0 = 2748_{(10)}$

```
h = 0xABC
print(h)
print(type(h))
```

```
2748
<class 'int'>
```

실수형 자료형(Floating-Point Type)

- 실수형 자료형 표현은 고정소수점과 부동소수점으로 구분

고정소수점(Fixed-Point)

- 기본적으로 고정된 자리수의 소수를 사용하는 고정소수점 방식
- 실수형 변수 `f`에 소수점이 포함된 `12.34` 값을 넣고 확인하기
- `type(f)`를 통해서 결과를 확인해보면 실수형(float) 자료형으로 출력
- 파이썬에서는 소수에서 정수부가 0인 경우에는 제외해서 표현 가능
- 코드에서 값 `.123`은 `0.123`을 의미

```
f = 12.34
print(f)
print(type(f))
```

```
12.34
<class 'float'>
```

```
f = .123
print(f)
```

```
0.123
```

부동소수점(Floating-Point)

- 컴퓨터에서 소수점의 위치가 고정되지 않고 넓은 범위의 값을 근사하여 표현하는 부동소수점 방식이 있음
- 부동소수점에 대한 개념을 모르는 경우에는 링크를 참고
- 파이썬에서는 e 기호를 이용하여 지수에 대한 표현을 사용
- `e` 변수에 할당한 `1234e-2` 라는 표현은 1234×10^{-2}를 의미하고 실제 결과 값은 12.34
- `type(e)` 함수를 통해 타입을 확인해보면 실수형
- 마찬가지로 `123e-3`은 123×10^{-3}를 의미하며 결과는 0.123

```
```

```
e = 1234e-2
print(e)
print(type(e))
```

```
12.34
<class 'float'>
```

```
e = 123e-3
print(e)
```

```
0.123
```

문자열 자료형(String Type)

- 문자열(string)은 문자들의 집합을 의미
- 문자열을 저장하는 변수를 문자열 변수라고 함

```
string = "Suan"
print(string)
print(type(string))
```

```
Suan
<class 'str'>
```

리스트 자료형(List Type)

- 리스트(list)는 여러 값을 하나의 변수에 담을 수 있는 자료형
- 대괄호 안에 여러 값을 쉽표를 구분자로 표현하여 리스트 변수 생성
- 간단하게 리스트 생성과 출력을 위해 변수 `l` 안에 값 1, 2, 3을 넣어보자
- 자료형을 출력해보면 'list'라는 결과 값을 가지는 것을 알 수 있음

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

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

- 리스트 자료형은 같은 자료형이 아니라 다른 자료형을 가지는 값들도 포함하여 구성 가능
- 숫자와 문자열이 포함된 리스트 변수 사용하기

```
l = [1, 'One', 2, 'Two', 3, 'Three']
print(l)
print(type(l))
```

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

튜플 자료형(Tuple Type)

- 튜플 자료형은 리스트 자료형과 유사하지만 생성, 삭제, 수정이 불가
- 튜플은 괄호 안에 여러 값을 쉼표로 구분하여 사용
- 변수 `t` 안에 값 1, 2, 3을 저장하고, 변수와 변수의 타입을 출력하기

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

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

범위 자료형(Range Type)

- 불변한 숫자 시퀀스 자료형

```
r = range(10)
print(r)
print(type(r))
```

```
range(0, 10)
<class 'range'>
```

집합 자료형(Set Type)

- 집합 자료형은 중복과 순서가 없다는 특징을 가지며 집합 처리를 쉽게할 수 있음
- 집합은 중괄호를 이용해서 값을 쉼표로 구분하여 넣음
- 변수 `s` 안에 값 2, 3, 1을 넣고 변수 `s`의 값과 타입을 출력한 예제
- 변경 불가능한 집합 자료형으로 fronzenset이 있음

```
s = {2, 3, 1}
print(s)
print(type(s))

fs = frozenset(s)
print(fs)
print(type(fs))
```

```
{1, 2, 3}
<class 'set'>
frozenset({1, 2, 3})
<class 'frozenset'>
```

딕셔너리 자료형(Dictionary Type)

- 딕셔너리 자료형은 다른 자료형과 달리 키(key)와 값(value)를 한 쌍으로 갖는 자료형
- 변수 `d`에 키와 값을 구분자 `:`를 이용해서 쌍으로 묶어서 `1: 'One', 2: 'Two'` 형태로 넣을 수 있음

```
d = {1: 'One', 2: 'Two'}
print(d)
print(type(d))
```

```
{1: 'One', 2: 'Two'}
<class 'dict'>
```

자료형 변환

- 파이썬에는 유용한 자료형을 제공하며, 자료형들간의 변환이 가능
- 자료형 변환에 사용되는 내장 함수(Built-in Function) 모음

함수	설명
book([x])	x를 불리언형으로 변환
int(x[,base])	x를 정수로 변환, x가 문자열일 경우 base 지정
float(x)	x를 실수형(부동소수점 숫자)로 변환
complex(real, [,imag])	복소수 생성
str(x)	객체 x를 문자열 표현으로 변환
repr(x)	객체 x를 표현식 문자열로 변환
eval(str)	문자열을 평가하고 객체를 반환
tuple(s)	s를 튜플로 변환
list(s)	s를 리스트로 변환
set(s)	s를 집합으로 변환
dict(d)	딕셔너리 생성, d는 (키 값) 튜플의 시퀀스이여야 함
frozenset(s)	s를 고정 집합으로 변환
chr(x)	정수를 문자로 변환
ord(x)	단일 문자를 정수 값으로 변환
bin(x)	정수를 2진수 문자열로 변환
oct(x)	정수를 8진수 문자열로 변환
hex(x)	정수를 16진수 문자열로 변환
bytes(x)	자료형을 바이트형으로 변환
bytearray(x)	자료형을 변경가능한 바이트형으로 변환
memoryview(x)	바이트 자료형을 메모리의 이진 데이터로 변환

자료형 계산

- 자료형들간의 계산이 가능하도록 몇 가지 유용한 내장 함수(Built-in Function) 제공

함수	설명
min(iterable)	두개 이상의 값에서 가장 작은 값을 반환

함수	설명
`max(iterable)`	두개 이상의 값에서 가장 큰 값을 반환
`sum(iterable)`	값들의 합을 반환
`divmod(a, b)`	`a` 를 `b` 로 나눈 값과 나머지를 쌍으로 반환
`abs(x)`	`x` 의 절대값을 반환
`pow(a, b)`	`a` 의 `b` 승의 값을 반환
`len(s)`	시퀀스(문자열, 바이트, 튜플, 리스트 등)의 갯수를 반환
`round(x)`	소수점 뒤를 반올림한 값을 반환
`all(iterable)`	모든 값이 `True` 이거나 비어있을 때 `True` 반환
`any(iterable)`	어떤 값이 `True` 이면 `True` 반환, 값이 비어있을 때 `False` 반환

연산자(Operators)

- 피연산자의 계산을 위해 다양한 연산자들이 존재
- 파이썬에서 제공하는 연산자들의 종류
    - 산술 연산자(Arithmetic Operators)
    - 비교 연산자(Comparison Operators)
    - 할당 연산자(Assignment Operators)
    - 비트 연산자(Bitwise Operators)
    - 논리 연산자(Logical Operators)
    - 멤버 연산자(Membership Operators)
    - 식별 연산자(Identity Operators)

산술 연산자(Arithmetic Operators)

연산자	설명	예제
+	덧셈	`c = a + b`
-	뺄셈	`c = a - b`
*	곱셈	`c = a * b`
/	나눗셈	`c = a / b`
%	나머지	`c = a % b`
**	제곱	`c = a ** b`
//	몫	`c = a // b`

```
a = 6
b = 4
print(a + b)
print(a - b)
print(a * b)
print(a / b)
```

```
print(a % b)
print(a ** b)
print(a // b)
```

```
10
2
24
1.5
2
1296
1
```

비교 연산자(Comparison Operators)

연산자	설명	예제
==	두 피연산자의 값이 동일하면 True	`a == b`
!=	두 피연산자의 값이 다르면 True	`a != b`
>	왼쪽 피연산자가 오른쪽 피연산자보다 크면 True	`a > b`
<	왼쪽 피연산자가 오른쪽 피연산자보다 작으면 True	`a < b`
>=	왼쪽 피연산자가 오른쪽 피연산자보다 크거나 같으면 True	`a >= b`
<=	왼쪽 피연산자가 오른쪽 피연산자보다 작거나 같으면 True	`a <= b`

```
a, b = 6, 4
print(a == b)
print(a != b)
print(a > b)
print(a < b)
print(a >= b)
print(a <= b)
```

```
False
True
True
False
True
False
```

할당 연산자(Assignment Operators)

연산자	설명	예제
=	오른쪽 피연산자의 값을 왼쪽 피연산자에 할당	`a, b = 20, 12`
+=	왼쪽 피연산자의 값을 오른쪽 피연산자에 더하고 그 결과를 왼쪽 피연산자에 대입	`a += b`
-=	왼쪽 피연산자의 값을 오른쪽 피연산자에 더하고 그 결과를 왼쪽 피연산자에 대입	`a -= b`
*=	왼쪽 피연산자의 값을 오른쪽 피연산자에 곱하고 그 결과를 왼쪽 피연산자에 대입	`a *= b`
/=	왼쪽 피연산자의 값을 오른쪽 피연산자에 나누고 그 결과를 왼쪽 피연산자에 대입	`a /= b`
%=	왼쪽 피연산자의 값을 오른쪽 피연산자에 나눈 나머지 값을 왼쪽 피연산자에 대입	`a %= b`
**=	왼쪽 피연산자의 값을 오른쪽 피연산자에 제곱한값을 왼쪽 피연산자에 대입	`a **= b`

연산자	설명	예제

```
a, b = 6, 4
a += b
print(a)
a -= b
print(a)
a *= b
print(a)
a /= b
print(a)
a %= b
print(a)
a **= b
print(a)
a //= b
print(a)
```

```
10
6
24
6.0
2.0
16.0
4.0
```

비트 연산자(Bitwise Operators)

연산자	설명	예제
&	두 피연산자의 비트를 AND 연산	`a & b`
|	두 피연산자의 비트를 OR 연산	`a | b`
^	두 피연산자의 비트를 XOR 연산	`a ^ b`
~	피연산자의 비트를 NOT 연산	`a ~ b`
<<	왼쪽 피연산자의 비트를 오른쪽 피연산자의 비트만큼 left shift 연산	`a << b`
>>	왼쪽 피연산자의 비트를 오른쪽 피연산자의 비트만큼 right shift 연산	`a >> b`

```
a, b = 6, 4
print(bin(a))
print(bin(b))
print(a & b, bin(a & b))
print(a | b, bin(a | b))
print(a ^ b, bin(a ^ b))
print(~a, bin(~a))
print(a << b, bin(a << b))
print(a >> b, bin(a >> b))
```

```
0b110
0b100
4 0b100
6 0b110
2 0b10
-7 -0b111
```

```
96 0b1100000
0 0b0
```

논리 연산자(Logical Operators)

연산자	설명	예제
and	두 피연산자가 모두 True이면 True	`True and True`
or	두 피연산자 중 하나라도 True이면 True	`True or False`
not	피연산자가 True이면 False, False이면 True	`not False`

```
print(True and True)
print(True and False)
print(True or False)
print(not True)
print(not False)
print(not (True and False))
```

```
True
False
True
False
True
True
```

멤버 연산자(Membership Operators)

연산자	설명	예제
in	멤버로 포함되어 있으면 True, 포함되어 있지 않으면 False	`a in l`
not in	멤버로 포함되어 있지 않으면 True, 포함되어 있으면 False	`a not in l`

```
a, b = 6, 4
l = [2, 4, 8]
print(a in l)
print(b in l)
print(a not in l)
print(b not in l)
```

```
False
True
True
False
```

식별 연산자(Identity Operators)

연산자	설명	예제
is	피연산자가 동일한 객체를 가리키면 True, 동일한 객체를 가리키고 있지 않으면 False	`a is b`
is not	피연산자가 동일한 객체를 가리키고 있지 않으면 True, 동일한 객체를 가리키면 False	`a is not b`

```
a, b =  6, 4
print(a is b)
print(a is not b)
a, b = 5, 5
print(a is b)
print(a is not b)
```

```
False
True
True
False
```

연산자 우선순위(Operators Precedence)

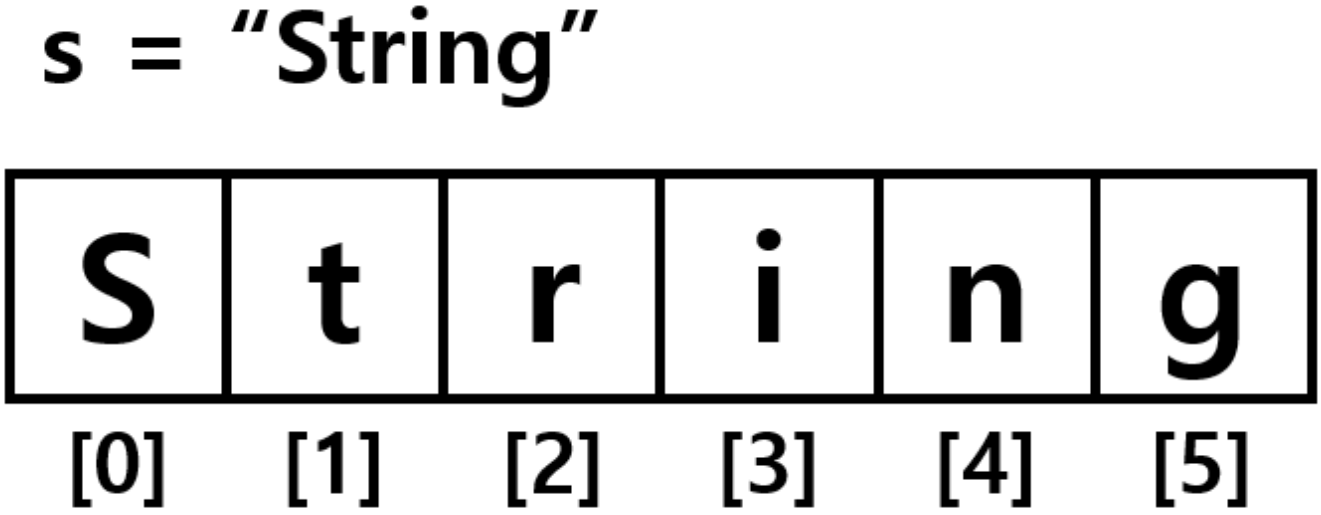
- 여러 연산자들이 사용되면 어떤 연산자들이 우선되는지를 결정하기 위해 연산자의 우선순위가 존재
- 연산자 중에서 괄호가 가장 높은 연산 순위를 가지며, 논리 연산자가 가장 낮은 연산 순위를 가짐

연산자	설명
()	괄호
**	지수(승수)
~, +, -	보수, 단항 덧셈과 뺄셈
*, /, %, //	곱셈, 나눗셈, 나머지, 몫
+, -	덧셈과 뺄셈
>>, <<	좌우 비트 시프트
&	비트 AND
^, |	비트 XOR, 비트 OR
<=, <>, >=	비교 연산자
==, !=	동등 연산자
=, %=, /=, //=, -=, +=, =, *=	할당 연산자
is, is not	식별 연산자
in, not in	멤버 연산자
not, or, and	논리 연산자

문자열(String)

문자열 정의

- 문자, 단어 등으로 구성된 문자들의 집합
- 시퀀스 자료형(sequence data type)




```
s = "String"
print(s)
print(s[0])
print(s[1])
print(s[2])
print(s[3])
print(s[4])
print(s[5])
```

```
String
S
t
r
i
n
g
```

```
s = "문자열"
print(s)
print(s[0])
print(s[1])
print(s[2])
```

```
문자열
문
자
열
```

문자열 생성

- 파이썬에서 문자열은 작은따옴표(') 또는 큰 따옴표(")로 표현

```
print('Hello')
print("파이썬은 재미있다.")
```

```
print('''파이썬은 심플하다.''')
print("""파이썬은 문자열 처리가 뛰어나다""")
```

```
Hello
파이썬은 재미있다.
파이썬은 심플하다.
파이썬은 문자열 처리가 뛰어나다
```

- 작은 따옴표 3개 또는 큰 따옴표 3개를 이용하여 여러 줄이 있는 문자열 생성

```
text = """동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산
대한 사람, 대한으로 길이 보전하세."""
print(text)
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세.
무궁화 삼천리 화려강산
대한 사람, 대한으로 길이 보전하세.
```

문자열 연산(String Operators)

문자열 더하기

- 연산자를 사용하여 문자열 연결

```
s1 = "동해물과 "
s2 = "백두산이"
print(s1 + s2)
```

```
동해물과 백두산이
```

문자열 곱하기

- * 연산자를 사용하여 문자열 반복

```
s = "String "
print(s * 3)
```

```
String String String
```

문자열 길이(length)

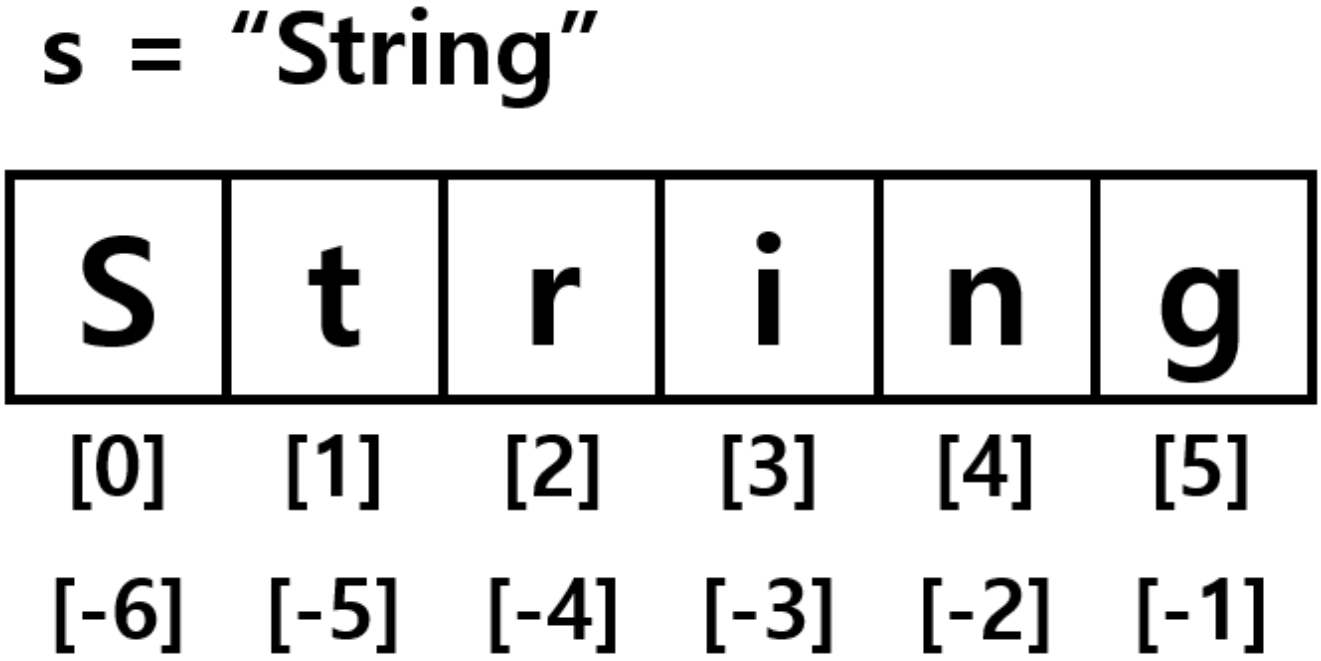
- 문자열 길이를 구하는 `len()` 함수

```
s = "String "
print(len(s))
print(len(s * 3))
```

```
7
21
```

문자열 인덱싱(indexing)

- 문자열은 리스트처럼 문자 하나하나가 상대적인 주소(offset)를 가짐
- 이 주소를 사용해 할당된 값을 가져오는 인덱싱을 사용




```
s = "String"
print(s)
print(s[-6])
print(s[-5])
print(s[-4])
print(s[-3])
print(s[-2])
print(s[-1])
```

```
String
S
t
r
i
n
g
```

문자열 슬라이싱(slicing)

- 문자열의 주소를 이용하여 문자열을 조각(부분)을 추출

```
s = "SuanLab - Suan Computer Laboratory"
print(s)
print(s[0:7])
print(s[:7])
print(s[10:])
print(s[-10:])
print(s[-19:-11])
```

```
SuanLab - Suan Computer Laboratory
SuanLab
SuanLab
Suan Computer Laboratory
Laboratory
Computer
```

문자열 메소드(String Methods)

- 파이썬은 문자열 처리를 아주 쉽게 처리할 수 있는 많은 메소드들을 제공
- 문자열을 다루는 주요 메소드들을 표로 정리

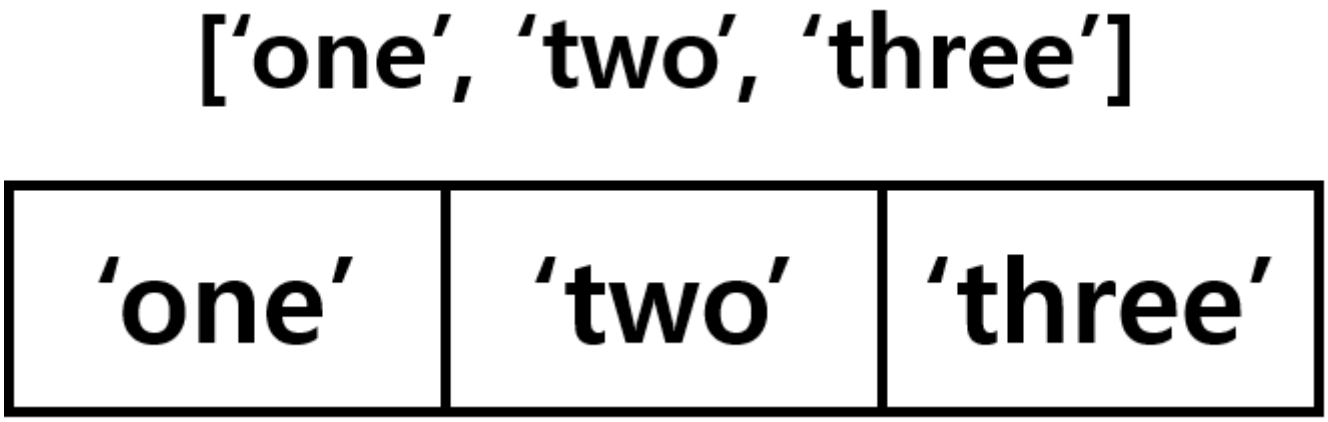
함수	설명
capitalize()	첫 문자를 대문자로하고, 나머지 문자를 소문자로 하는 문자열 반환
casefold()	모든 대소문자 구분을 제거
count(sub, [, start[, end]])	[start, end] 범위에서 부분 문자열 sub의 중복되지 않은 수를 반환
find(sub [, start [, end]])	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 인덱스를 반환. sub가 발견되지 않는
rfind(sub [, start [, end]])	[start, end]에서 부분 문자열 sub가 문자열의 가장 작은 큰 인덱스를 반환. sub가 발견되지 않
index(sub [, start [, end]])	find()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
rindex(sub [, start [, end]])	rfind()과 유사하지만 부분 문자열 sub가 없으면 ValueError 발생
isalnum()	문자열의 모든 문자가 영숫자로 1개 이상 있으면 True, 아니면 False 반환
isalpha()	문자열의 모든 문자가 영문자로 1개 이상 있으면 True, 아니면 False 반환
isdecimal()	문자열의 모든 문자가 10진수 문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isdigit()	문자열의 모든 문자가 숫자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isnumeric()	문자열의 모든 문자가 수치형이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isidentifier()	문자열이 유효한 식별자인 경우 True 반환
isspace()	문자열 내에 공백 문자가 있고, 문자가 1개 이상 있으면 True, 그렇지 않으면 False
istitle()	문자열이 제목이 있는 문자열에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
islower()	문자열의 모든 문자가 소문자이며 1개 이상 있을 때 True, 그렇지 않으면 False 반환
isupper()	문자열의 문자가 모두 대문자에 문자가 1개 이상 있으면 True, 그렇지 않으면 False
join(iterable)	iterable에 있는 문자열에 연결된 문자열을 반환

함수	설명
center(width [, fillchar])	길이 너비만큼 중앙정렬된 문자열 반환
ljust(width [, fillchar])	너미만큼의 문자열에서 왼쪽 정렬된 문자열을 반환
rjust(width [, fillchar])	너미만큼의 문자열에서 오른쪽 정렬된 문자열을 반환
lower()	모든 대소문자가 소문자로 변환된 문자열을 반환
upper()	문자열에서 모든 문자를 대문자로 변환한 문자열을 반환
title()	문자열에서 첫 글자만 대문자이고 나머지는 소문자인 문자열 반환
swapcase()	문자열에서 소문자를 대문자로 대문자를 소문자로 변환한 문자열 반환
strip([chars])	문자열 양쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
lstrip([chars])	문자열 왼쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
rstrip([chars])	문자열 오른쪽에 지정된 chars 또는 공백을 제거한 문자열을 반환
partition(sep)	문자열에서 첫번째 sep를 기준으로 분할하여 3개의 튜플을 반환
rpartition(sep)	문자열에서 마지막 sep를 기준으로 분할하여 3개의 튜플을 반환
replace(old, new[,count])	문자열의 모든 old를 new로 교체한 문자열을 반환
split(sep=None, maxsplit=1)	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
rsplit(sep=None, maxsplit=1)	sep를 구분자 문자열로 사용하여 문자열의 단어 목록을 반환
splitlines([keepends])	문자열에서 라인 단위로 구분하여 리스트를 반환
startswith(prefix [, start[, end]])	[start, end] 범위에서 지정한 prefix로 시작하면 True, 아니면 False 반환
endswith(suffix [, start[, end]])	[start, end] 범위에서 지정한 suffix로 끝나면 True, 아니면 False 반환
zfill(width)	너비 만큼의 문자열에서 비어있는 부분에 '0'이 채워진 문자열 반환

리스트, 튜플, 세트, 딕셔너리

리스트(List)

- 변경 가능한 시퀀스 자료형
- 하나의 변수에 여러 값 할당 가능
- 같은 자료형이 아니라 다른 자료형을 가지는 값들을 포함 가능
- 일반적으로 비슷한 항목들의 모음을 순서대로 저장하는데 사용

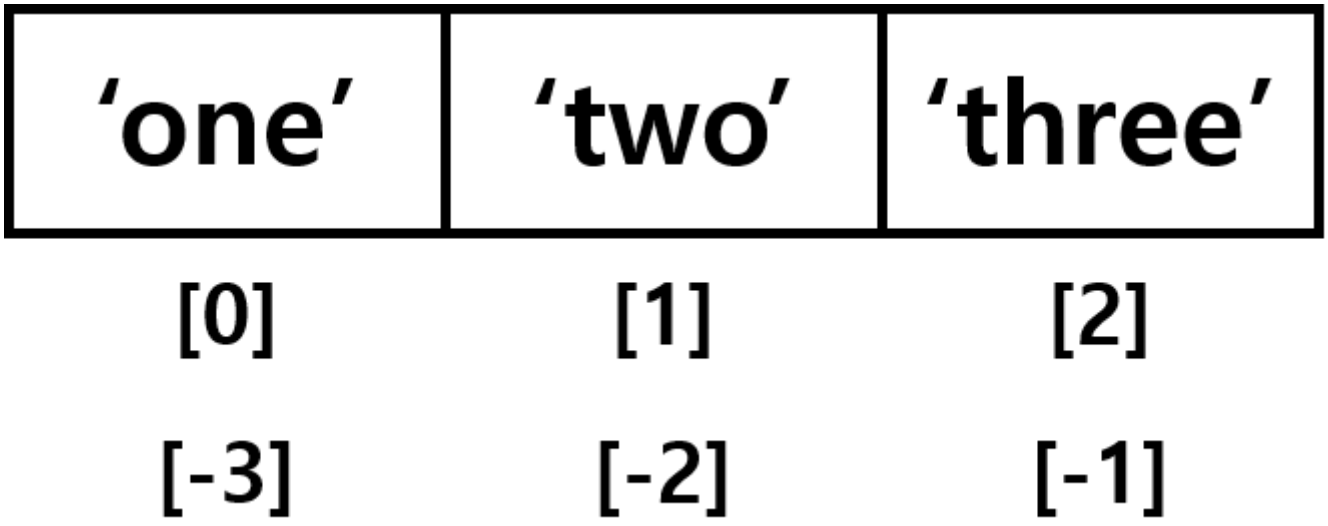



```
print([])
print([1, 2, 3])
print(['One', 'Two', 'Three'])
print([1, 'One', 2, 'Two', 3, 'Three'])
print([1, 2, 3, ['One', 'Two', 'Three']])
```

```
[]
[1, 2, 3]
['One', 'Two', 'Three']
[1, 'One', 2, 'Two', 3, 'Three']
[1, 2, 3, ['One', 'Two', 'Three']]
```

리스트 인덱싱(List Indexing)

- 리스트의 각 위치에 해당하는 값에 접근하기 위해서 주소 개념의 숫자를 사용



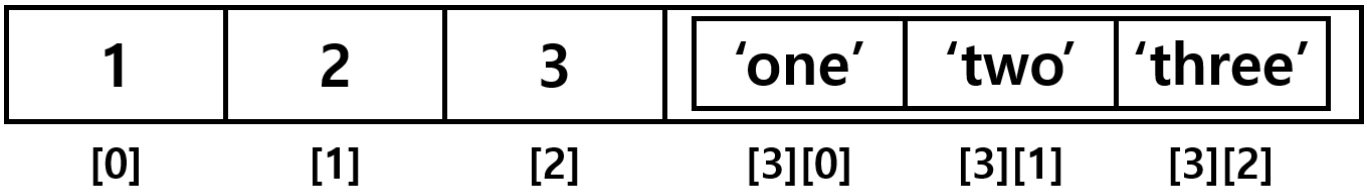

```
list = ['One', 'Two', 'Three']
print(list)
print(list[0])
print(list[1])
print(list[2])
print(list[-1])
```

```
print(list[-2])
print(list[-3])
```

```
['One', 'Two', 'Three']
One
Two
Three
Three
Two
One
```

중첩 리스트 인덱싱(Nested List Indexing)

- 리스트 안에 리스트가 있을 경우 인덱스 접근 방법




```
list = [1, 2, 3, ['One', 'Two', 'Three']]
print(list)
print(list[3])
print(list[3][0])
print(list[3][1])
print(list[3][2])
```

```
[1, 2, 3, ['One', 'Two', 'Three']]
['One', 'Two', 'Three']
One
Two
Three
```

리스트 슬라이싱(List Slicing)

- 리스트의 인덱스를 통해서 부분만 가져올 때 사용

```
list = ['One', 'Two', 'Three']
print(list)
print(list[0:])
print(list[1:2])
print(list[1:])
```

```
['One', 'Two', 'Three']
['One', 'Two', 'Three']
['Two']
['Two', 'Three']
```

중첩 리스트 슬라이싱(Nested List Slicing)

- 중첩된 리스트에서 일부분만 인덱스를 통해서 가져올 때 사용

```
list = [1, 2, 3, ['One', 'Two', 'Three']]
print(list)
print(list[2:4])
print(list[3][2:])
print(list[3][:2])
```

```
[1, 2, 3, ['One', 'Two', 'Three']]
[3, ['One', 'Two', 'Three']]
['Three']
['One', 'Two']
```

리스트 연산자/함수(List Operators/Function)

- 더하기 + 연산자
- 곱하기 * 연산자
- 리스트 길이를 구하는 `len()` 함수

```
list_1 = ['One', 'Two', 'Three']
list_2 = ['Four', 'Five', 'Six']
print(list_1)
print(list_2)
print(list_1 + list_2)
print(list_1 * 3)
print(len(list_1))
print(len(list_1 * 3))
```

```
['One', 'Two', 'Three']
['Four', 'Five', 'Six']
['One', 'Two', 'Three', 'Four', 'Five', 'Six']
['One', 'Two', 'Three', 'One', 'Two', 'Three', 'One', 'Two', 'Three']
3
9
```

리스트 수정(List Modify)

- 리스트 인덱스 주소를 이용한 값 수정

```
list = ['One', 'Two', 'Three']
print(list)
list[2] = 3
print(list)
list[1] = 2
print(list)
list[0] = 1
print(list)
```

```
['One', 'Two', 'Three']
```

```
['One', 'Two', 3]
['One', 2, 3]
[1, 2, 3]
```

리스트 메소드(List Methods)

append()

- 리스트 요소 추가

```
list = ['One', 'Two', 'Three']
list.append('Four')
print(list)
list.append([1, 2, 3, 4])
print(list)
```

```
['One', 'Two', 'Three', 'Four']
['One', 'Two', 'Three', 'Four', [1, 2, 3, 4]]
```

sort()

- 리스트 정렬

```
list = [10, 40, 20, 30]
print(list)
list.sort()
print(list)
list = ['orange', 'apple', 'banana', 'strawberry']
print(list)
list.sort()
print(list)
```

```
[10, 40, 20, 30]
[10, 20, 30, 40]
['orange', 'apple', 'banana', 'strawberry']
['apple', 'banana', 'orange', 'strawberry']
```

reverse()

- 리스트 요소 반전

```
list = ['orange', 'apple', 'banana', 'strawberry']
print(list)
list.reverse()
print(list)
```

```
['orange', 'apple', 'banana', 'strawberry']
['strawberry', 'banana', 'apple', 'orange']
```

index()

- 리스트의 요소 값에 대한 인덱스를 반환

```
list = [10, 40, 20, 30]
print(list)
print(list.index(10))
print(list.index(20))
```

```
[10, 40, 20, 30]
0
2
```

insert()

- 리스트 요소 삽입

```
list = [10, 40, 20, 30]
print(list)
list.insert(4, 50)
print(list)
list.insert(0, 60)
print(list)
```

```
[10, 40, 20, 30]
[10, 40, 20, 30, 50]
[60, 10, 40, 20, 30, 50]
```

remove()

- 리스트 요소 제거

```
list = [10, 40, 20, 30]
print(list)
list.remove(40)
print(list)
list.remove(30)
print(list)
```

```
[10, 40, 20, 30]
[10, 20, 30]
[10, 20]
```

del

- 리스트 요소 제거 연산

```
list = [10, 40, 20, 30]
print(list)
```

```
del list[0]
print(list)
del list[2]
print(list)
```

```
[10, 40, 20, 30]
[40, 20, 30]
[40, 20]
```

pop()

- 리스트 요소를 방출

```
list = [10, 40, 20, 30]
print(list)
list.pop()
print(list)
list.pop(0)
print(list)
```

```
[10, 40, 20, 30]
[10, 40, 20]
[40, 20]
```

count()

- 리스트 요소의 갯수 계산

```
list = [10, 20, 20, 30, 30, 30]
print(list)
print(list.count(30))
print(list.count(20))
```

```
[10, 20, 20, 30, 30, 30]
3
2
```

extend()

- 리스트 확장

```
list = [10, 40, 20, 30]
print(list)
list.extend([50, 60])
print(list)
```

```
[10, 40, 20, 30]
[10, 40, 20, 30, 50, 60]
```

튜플(Tuple)

- 리스트와 유사하지만 변경 불가능한 시퀀스 자료형
- 하나의 변수에 여러 값 할당 가능
- '('와 ')'를 사용하여 표현

```
print(())
print((1, 2, 3))
print(('One', 'Two', 'Three'))
print((1, 'One', 2, 'Two', 3, 'Three'))
print((1, 2, 3, ('One', 'Two', 'Three')))
```

```
()
(1, 2, 3)
('One', 'Two', 'Three')
(1, 'One', 2, 'Two', 3, 'Three')
(1, 2, 3, ('One', 'Two', 'Three'))
```

튜플 인덱싱(Tuple Indexing)

- 튜플의 각 위치에 해당하는 값에 접근하기 위해서 주소 개념의 숫자를 사용

```
tuple = ('One', 'Two', 'Three')
print(tuple)
print(tuple[0])
print(tuple[-1])
```

```
('One', 'Two', 'Three')
One
Three
```

중첩 튜플 인덱싱(Nested Tuple Indexing)

- 튜플 안에 튜플이 중첩되어 있을 경우, 인덱스 접근 방법

```
tuple = (1, 2, 3, ('One', 'Two', 'Three'))
print(tuple)
print(tuple[3])
print(tuple[3][1])
```

```
(1, 2, 3, ('One', 'Two', 'Three'))
('One', 'Two', 'Three')
Two
```

튜플 슬라이싱(Tuple Slicing)

- 튜플의 인덱스를 통해서 부분만 가져올 때 사용

```
tuple = ('One', 'Two', 'Three')
print(tuple)
print(tuple[0:])
print(tuple[1:2])
print(tuple[1:])
```

```
('One', 'Two', 'Three')
('One', 'Two', 'Three')
('Two',)
('Two', 'Three')
```

중첩 튜플 슬라이싱(Nested Tuple Slicing)

- 중첩된 튜플에서 일부분만 인덱스를 통해서 가져올 때 사용

```
tuple = (1, 2, 3, ('One', 'Two', 'Three'))
print(tuple)
print(tuple[3])
print(tuple[3][2:])
print(tuple[3][:2])
```

```
(1, 2, 3, ('One', 'Two', 'Three'))
('One', 'Two', 'Three')
('Three',)
('One', 'Two')
```

세트(Set)

- 데이터 중복을 허용하지 않는 구조
- 순서가 없는 데이터 집합을 위한 구조
- 인덱싱으로 값을 접근할 수 없음

```
print({})
print({'One', 'Two', 'Three'})
print({10, 20, 30, 40})
```

```
{}
{'Two', 'Three', 'One'}
{40, 10, 20, 30}
```

세트 연산자(Set Operators)

- 교집합: &
- 합집합: |
- 차집합: -
- 여집합: ^

```
set_1 = {10, 20, 20, 30}
set_2 = {30, 30, 40, 50}
print(set_1)
print(set_2)
print(set_1 & set_2)
print(set_1 | set_2)
print(set_1 - set_2)
print(set_1 ^ set_2)
```

```
{10, 20, 30}
{40, 50, 30}
{30}
{50, 20, 40, 10, 30}
{10, 20}
{40, 10, 50, 20}
```

세트 메소드(Set Methods)

- 교집합: intersection()
- 합집합: union()
- 차집합: difference()
- 여집합: symmetric_difference()

```
set_1 = {10, 20, 20, 30}
set_2 = {30, 30, 40, 50}
print(set_1)
print(set_2)
print(set_1.intersection(set_2))
print(set_1.union(set_2))
print(set_1.difference(set_2))
print(set_1.symmetric_difference(set_2))
```

```
{10, 20, 30}
{40, 50, 30}
{30}
{50, 20, 40, 10, 30}
{10, 20}
{40, 10, 50, 20}
```

- 요소 추가: add()
- 여러 요소 추가: update()
- 요소 제거: remove()
- 요소 제거: discard()
- 모든 요소 제거: clear()

```
set = {10, 20, 30, 40}
print(set)

set.add(50)
```

```
print(set)

set.update([60, 70])
print(set)

set.remove(70)
set.remove(60)
print(set)

set.discard(30)
print(set)

set.clear()
print(set)
```

```
{40, 10, 20, 30}
{40, 10, 50, 20, 30}
{70, 40, 10, 50, 20, 60, 30}
{40, 10, 50, 20, 30}
{40, 10, 50, 20}
set()
```

딕셔너리(Dictionary)

- 키(key)와 값(value)의 쌍으로 구성된 데이터
- 순서가 없는 데이터
- 키를 통해 값을 얻음
- 동일한 키가 있을경우 덮어씀

```
dic = {1:'One', 2:'Two', 3:'Three'}
print(dic)

dic = {1:'One', 2:'Two', 3:'Three', 1:'One', 2:'Two', 3:'Three'}
print(dic)
```

```
{1: 'One', 2: 'Two', 3: 'Three'}
{1: 'One', 2: 'Two', 3: 'Three'}
```

딕셔너리 요소 추가/삭제

- 딕셔너리의 해당 키 값에 값을 추가하여 요소 추가
- del을 이용하여 요소 제거

```
dic = {1:'One', 2:'Two', 3:'Three'}
print(dic)
print(dic[2])
dic[4] = 'Four'
```

```
print(dic)
dic[5] = 'Five'
print(dic)
del dic[4]
print(dic)
```

```
{1: 'One', 2: 'Two', 3: 'Three'}
Two
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four'}
{1: 'One', 2: 'Two', 3: 'Three', 4: 'Four', 5: 'Five'}
{1: 'One', 2: 'Two', 3: 'Three', 5: 'Five'}
```

딕셔너리 메소드(Dictionary Methods)

메소드	설명
keys()	딕셔너리의 키 가져오기
values()	딕셔너리의 값 가져오기
items()	딕셔너리의 키와 값을 모두 가져오기
get()	딕셔너리에서 키에 해당하는 값 가져오기
pop()	딕셔너리에서 키에 해당하는 값 추출하기
clear()	딕셔너리의 모든 요소를 제거하기

```
dic = {1:'One', 2:'Two', 3:'Three'}
print(dic)
print(dic.keys())
print(dic.values())
print(dic.items())
print(dic.get(2))
print(dic.pop(3))
print(dic)
dic.clear()
print(dic)
```

```
{1: 'One', 2: 'Two', 3: 'Three'}
dict_keys([1, 2, 3])
dict_values(['One', 'Two', 'Three'])
dict_items([(1, 'One'), (2, 'Two'), (3, 'Three')])
Two
Three
{1: 'One', 2: 'Two'}
{}
```

제어문(Control Statement)

조건문(Conditional Statement)

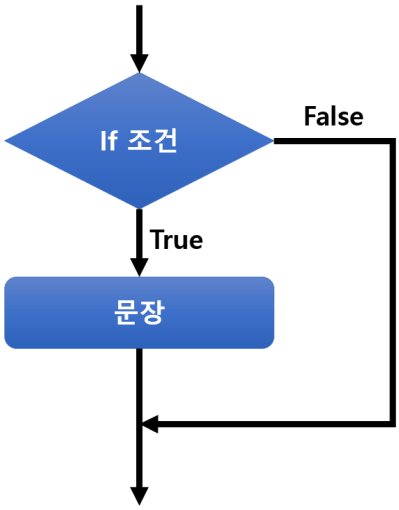
- 조건에 따라 문장을 수행
- 주어진 조건을 판단하고 상황에 맞는 처리가 필요할 때 사용
- 파이썬에서 제공하는 조건문
    - if
    - else
    - elif

if 문

- if 문은 True와 False를 판단하는 조건문
- if 조건 뒤에는 콜론(:)
- if 기본 문법

```
if <조건>:
  <문장>
```

- if 구조도



- if 문을 이용한 미세먼지 측정
    - 미세먼지 농도 pm
    - 35 초과는 미세먼지 농도 나쁨

```
pm = 40
if pm > 35:
  print("미세먼지 농도: 나쁨")
```

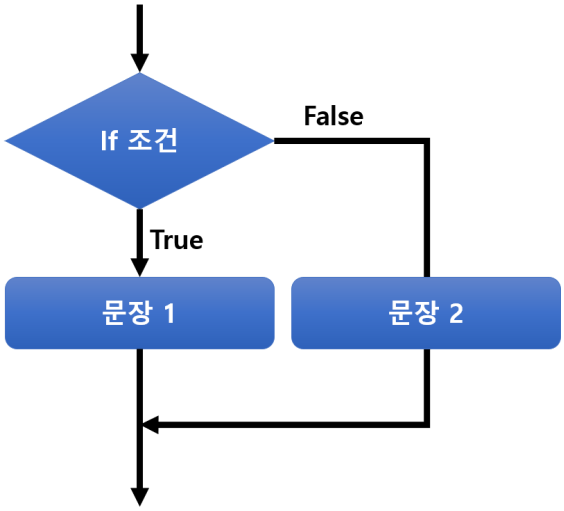
```
미세먼지 농도: 나쁨
```

if-else 문

- if-else 기본 문법

```
if <조건>:
  <문장 1>
else:
  <문장 2>
```

- else 문 뒤에는 콜론(:)
- if-else 구조도



- if-else 문을 이용한 미세먼지 측정
  - 미세먼지 농도 pm
  - 36 이상은 미세먼지 농도 나쁨
  - 35 이하는 미세먼지 농도 좋음

```
pm = 30
if pm >= 36:
  print("미세먼지 농도: 나쁨")
else:
  print("미세먼지 농도: 좋음")
```

```
미세먼지 농도: 좋음
```

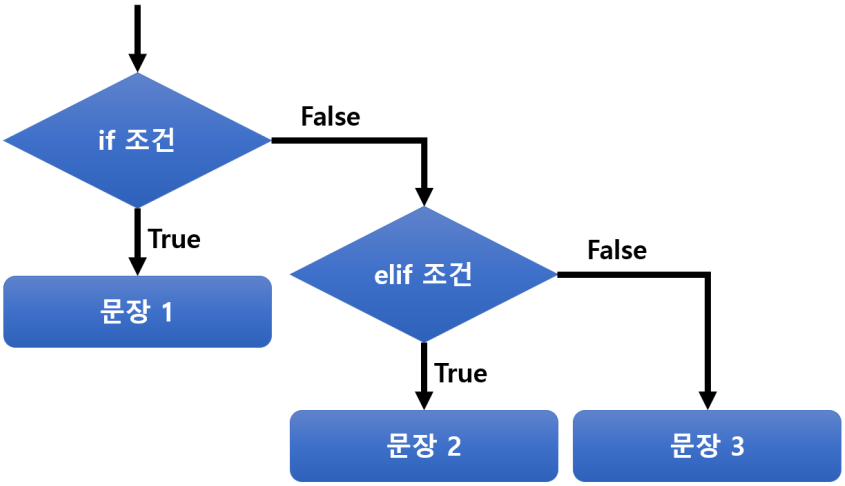
if-elif-else 문

- if-elif-else 기본 문법

```
if <조건>:
  <문장 1>
elif <조건>:
  <문장 2>
```

```
else:
  <문장 3>
```

- elif 문 조건 뒤에는 콜론(:)
- if-elif-else 구조도



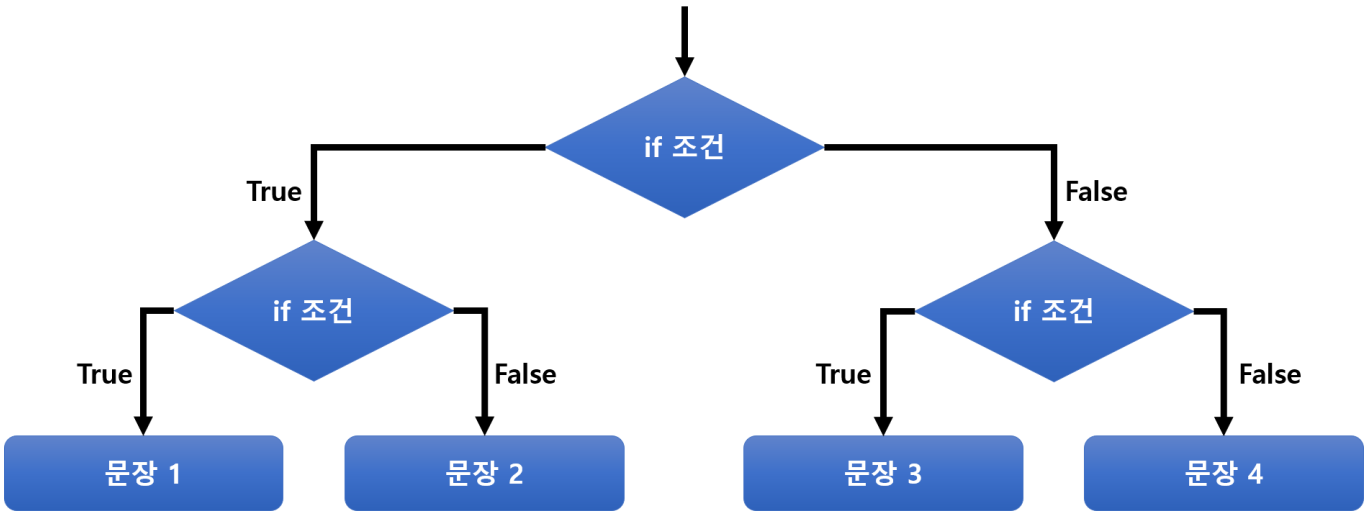
- if-elif-else 문을 이용한 미세먼지 측정
  - 미세먼지 농도 pm
  - 미세먼지 농도 0~15: 좋음
  - 미세먼지 농도 16~35: 보통
  - 미세먼지 농도 36~75: 나쁨
  - 미세먼지 농도 76~: 매우나쁨

```
pm = 40
if pm < 16:
  print("미세먼지 농도: 좋음")
elif pm < 36:
  print("미세먼지 농도: 보통")
elif pm < 76:
  print("미세먼지 농도: 나쁨")
else:
  print("미세먼지 농도: 매우나쁨")
```

```
미세먼지 농도: 나쁨
```

중첩 if 문

- if 문 안에 if 문에 포함된 형태



- 중첩 if 문을 이용한 미세먼지 측정
  - 미세먼지 농도 pm
  - 미세먼지 농도 0~15: 좋음
  - 미세먼지 농도 16~35: 보통
  - 미세먼지 농도 36~75: 나쁨
  - 미세먼지 농도 76~: 매우나쁨

```
pm = 80
if pm < 36:
  if pm < 16:
    print("미세먼지 농도: 좋음")
  else:
    print("미세먼지 농도: 보통")
else:
  if pm < 76:
    print("미세먼지 농도: 나쁨")
  else:
    print("미세먼지 농도: 매우나쁨")
```

```
미세먼지 농도: 매우나쁨
```

if-pass 문

- 조건문은 있지만 실행할 문장이 없는 경우, 오류가 발생하지 않도록 무시하고 넘어가는 기능

```
if 10 > 5:
  print(10)
else:
  pass
```

```
10
```

if 조건 연산자

- 비교연산자: <, >, ==, !=, >=, <=

```
if 2 > 1:
  print(2)

if 3 == 3:
  print(3)

if 1 != 2:
  print(1)
```

```
2
3
1
```

- 논리연산자: and, or, not

```
rain = True
snow = True
sun = False

if rain and snow:
  print("진눈깨비")

if not sun:
  print("흐림")
else:
  print("맑음")
```

```
진눈깨비
흐림
```

- 멤버연산자: in, not in

```
list = ['One', 'Two', 'Three']

if 'One' in list:
  print('One')

if 'Four' not in list:
  print('No')
```

```
One
No
```

- 식별연산자: is, is not

```
if 'One' is 'One':
  print('One')
```

```
if 'One' is not 'Two':
  print('One is not Two')
```

```
One
One is not Two
```

조건부 표현식(Conditional Expression)

- 한 라인으로 조건식을 사용한 표현

```
score = 75
msg = "통과" if score >= 70 else "탈락"
print(msg)
```

```
통과
```

반복문(Repetitive Statement)

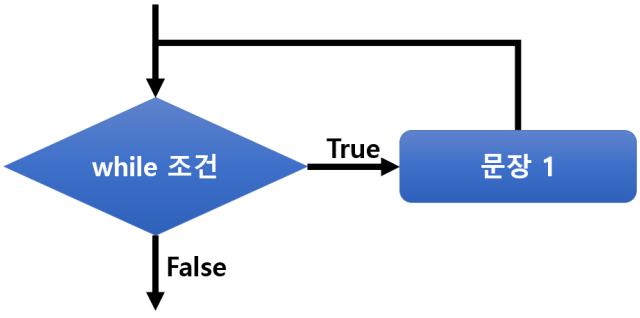
- 문장을 반복적으로 수행
- 정해진 동작을 반복하여 처리할 때 사용
- 파이썬에서 제공하는 반복문
  - while
  - for

while 문

- 어떤 조건이 만족하는 동안 문장을 수행하고 만족하지 않는 경우 수행 중단
- while 문 기본 문법

```
while <조건>:
  <문장>
```

- while 문 구조도



- while 문 예제: 1부터 10까지 반복

```
i = 1
while i <= 10:
  print(i)
  i += 1
```

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

- while 문 예제: 1부터 10까지 더하기

```
i = 1
sum = 0
while i <= 10:
  sum += i
  i += 1

print(sum)
```

```
55
```

for 문

- 반복 범위를 지정하여 반복 수행
- for 문 기본 문법

```
for 변수 in 리스트, 튜플, 문자열:
  <문장>
```

- 리스트 요소 반복

```
list = ['One', 'Two', 'Three']
for i in list:
  print(i)
```

```
One
Two
Three
```

- 튜플 요소 반복

```
tuple = ('One', 'Two', 'Three')
for i in tuple:
  print(i)
```

```
One
Two
Three
```

- 문자열 요소 반복

```
string = "SuanLab"
for i in string:
  print(i)
```

```
S
u
a
n
L
a
b
```

▼ range()

- 범위 반복에 사용
- range 문법

```
for 변수 in range(시작값, 마지막값, 증가값):
  <문장>
```

- 시작값과 증가값은 생략 가능
- 생략할 때 시작값은 0, 증가값은 1

```
sum = 0
for i in range(101):
  sum += i

print(sum)
```

```
5050
```

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

```
1
```

```
3
5
7
9
```

- 범위 반복 range를 이용한 구구단

```
for i in range(2, 10):
  for j in range(1, 10):
    print('{0} x {1} = {2}'.format(i, j, i * j))
```

```
2 x 1 = 2
2 x 2 = 4
2 x 3 = 6
2 x 4 = 8
2 x 5 = 10
2 x 6 = 12
2 x 7 = 14
2 x 8 = 16
2 x 9 = 18
3 x 1 = 3
3 x 2 = 6
3 x 3 = 9
3 x 4 = 12
3 x 5 = 15
3 x 6 = 18
3 x 7 = 21
3 x 8 = 24
3 x 9 = 27
4 x 1 = 4
4 x 2 = 8
4 x 3 = 12
4 x 4 = 16
4 x 5 = 20
4 x 6 = 24
4 x 7 = 28
4 x 8 = 32
4 x 9 = 36
5 x 1 = 5
5 x 2 = 10
5 x 3 = 15
5 x 4 = 20
5 x 5 = 25
5 x 6 = 30
5 x 7 = 35
5 x 8 = 40
5 x 9 = 45
6 x 1 = 6
6 x 2 = 12
6 x 3 = 18
6 x 4 = 24
6 x 5 = 30
6 x 6 = 36
6 x 7 = 42
6 x 8 = 48
6 x 9 = 54
7 x 1 = 7
```

```
7 x 2 = 14
7 x 3 = 21
7 x 4 = 28
7 x 5 = 35
7 x 6 = 42
7 x 7 = 49
7 x 8 = 56
7 x 9 = 63
8 x 1 = 8
8 x 2 = 16
8 x 3 = 24
```

_ 기능

- for문을 사용하면서 iterator 역할을 위해서 i 변수가 필요
- for문 이후에는 iterator 변수 i가 필요하지 않음
- 이후에 사용되지 않을 변수에 어떤 이름을 부여하고 싶지 않을 때 _ 를 사용

```
for i in range(5):
  print("SuanLab")
```

```
SuanLab
SuanLab
SuanLab
SuanLab
SuanLab
```

```
for _ in range(5):
  print("SuanLab")
```

```
SuanLab
SuanLab
SuanLab
SuanLab
SuanLab
```

else 문

- 반복이 종료된 후에 한번 더 실행되는 문장

```
i = 1
sum = 0
while i <= 10:
  sum += i
  i += 1
else:
  print(sum)
```

```
55
```

```
sum = 0
```

```
for i in range(11):
  sum += i
else:
  print(sum)
```

```
55
```

break 문

- 반복문 종료

```
i = 0
while i < 100:
  print(i)
  if i == 10:
    break
  i += 1
```

```
0
1
2
3
4
5
6
7
8
9
10
```

```
for i in range(100):
  print(i)
  if i == 10:
    break
```

```
0
1
2
3
4
5
6
7
8
9
10
```

continue 문

- 반복 조건문으로 이동

```
i = 0
while i < 10:
```

```
  i += 1
  if i % 2 == 0:
    continue
  print(i)
```

```
1
3
5
7
9
```

```
for i in range(10):
  if i % 2 == 0:
    continue
  print(i)
```

```
1
3
5
7
9
```

리스트 내포(List Comprehension)

- 리스트 안에 for 문과 if 문 사용

```
list = [1, 2, 3, 4, 5]
print([i * 2 for i in list])
print([i * 2 for i in range(10) if i % 2 == 0])
```

```
[2, 4, 6, 8, 10]
[0, 4, 8, 12, 16]
```

입력과 출력

표준 입출력(Standard Input/Output)

표준 입력 함수

- `input()` 함수: 콘솔 창을 통해서 사용자 입력을 받는 표준 입력 함수
- `input()` 함수 안에 입력 받기 위한 질문을 텍스트로 넣을 수 있음

```
name = input("이름: ")
print(name)
```

```
이름: 이수안
이수안
```

표준 출력 함수

- `print()` 함수: 콘솔 창을 통해서 결과를 출력하는 표준 출력 함수
- sep='separator': 개체가 분리하는 방법 지정, 기본값은 ' '
- end='end': 끝에 출력할 항목 지정, 기본값은 '\n'
- file: 쓰기 방법이 있는 객체, 기본값은 sys.stdout
- flush: True일 경우 flush하고, False일 경우 버퍼, 기본값은 False

```
print("Hello Python")
print("안녕 파이썬")
print(7)
print(3.14)
print([1, 2, 3])

print("Hello", "Python", sep = "---")
print(1, 2, 3)
print(1, end=' ')
print(2, end=' ')
print(3, end=' ')
```

```
Hello Python
안녕 파이썬
7
3.14
[1, 2, 3]
Hello---Python
1 2 3
1 2 3
```

파일 입출력(File Input/Output)

파일 입출력 과정

- 파일의 입출력 과정은 단계를 가짐
- 파일을 열고, 파일을 읽거나 쓰고, 파일을 닫는 순서

파일 열기/닫기

- file.txt 파일을 생성하고, 열고 닫기

```
f = open("file.txt", 'w')
f = open("file.txt", 'r')
f.close()
```

```
!ls file.txt
```

```
file.txt
```

파일 모드(File Mode)

파일 모드	설명
(생략)	r과 동일한 모드
r	읽기 모드(기본값)
w	쓰기 모드, 기존에 파일이 있으면 덮어쓰기
a	쓰기 모드, 기존에 파일이 있으면 이어서 쓰기(append)
+	읽기/쓰기 모드
t	텍스트 모드(기본값), 텍스트 파일을 처리
b	이진 모드, 이진 파일을 처리

텍스트 파일 쓰기

write()

- `write()`를 이용하여 파일에 쓰기

```
f = open("file.txt", 'w')
f.write("Hello Python")
f.close()
```

```
!cat file.txt
```

```
Hello Python
```

writelines()

- `writelines()`를 이용하여 list를 파일에 쓰기

```
list = ["One₩n", "Two₩n", "Three₩n"]
f = open("file.txt", 'w')
f.writelines(list)
f.close()
```

```
!cat file.txt
```

```
One
Two
Three
```

표준 입력 → 파일 쓰기

- 표준 입력을 `input()` 함수를 통해 입력 받고, `write()` 함수를 통해 파일에 쓰기

```
text = input("입력: ")
f = open("file.txt", 'w')
f.write(text)
f.close()
```

```
입력: 이수안
```

```
!cat file.txt
```

이수안

```
text = [str(text + '₩n') for text in input("여러 값 입력: ").split()]
f = open("file.txt", 'w')
f.writelines(text)
f.close()
```

```
여러 값 입력: 1 2 3 4 5
```

```
!cat file.txt
```

```
1
2
3
4
5
1
2
3
4
5
q
```

```
x = ""
text = ""
while x != 'q':
  x = input("반복 입력: ")
  text += x + '₩n'

f = open("file.txt", 'w')
f.writelines(text)
f.close()
```

```
반복 입력: 1
반복 입력: 2
반복 입력: 3
반복 입력: 4
반복 입력: 5
반복 입력: q
```

```
!cat file.txt
```

```
1
2
3
4
5
q
```

텍스트 파일 출력

- 텍스트 파일 예제

```
!echo "동해물과 백두산이 마르고 닳도록" > anthem.txt
!echo "하느님이 보우하사 우리나라 만세" >> anthem.txt
```

readline()

- `readline()`를 이용하여 텍스트 파일을 라인 단위로 읽고 화면에 출력

```
f = open('anthem.txt', 'r')
while True:
  line = f.readline()
  if not line: break
  print(line)

f.close()
```

```
동해물과 백두산이 마르고 닳도록

하느님이 보우하사 우리나라 만세
```

readlines()

- `readlines()`를 이용하여 텍스트 파일의 여러 라인을 읽고 화면에 출력

```
f = open("anthem.txt", 'r')
lines = f.readlines()
print(lines)
f.close()
```

```
['동해물과 백두산이 마르고 닳도록₩n', '하느님이 보우하사 우리나라 만세₩n']
```

read()

- `read()`를 이용하여 텍스트 파일을 읽고 화면에 출력

```
f = open("anthem.txt", 'r')
data = f.read()
```

```
print(data)
f.close()
```

```
동해물과 백두산이 마르고 닳도록
하느님이 보우하사 우리나라 만세
```

tell()

- 파일 포인터 위치 변경

```
f = open("anthem.txt", 'r')
while True:
  print(f.tell())
  line = f.readline()
  if not line: break
  print(line)

f.close()
```

```
0
동해물과 백두산이 마르고 닳도록

46
하느님이 보우하사 우리나라 만세

92
```

seek()

- 파일 포인터 위치 이동

```
f = open("anthem.txt", 'r')
f.seek(46)
line = f.readline()
print(line)
f.seek(0)
line = f.readline()
print(line)
f.close()
```

```
하느님이 보우하사 우리나라 만세

동해물과 백두산이 마르고 닳도록
```

with 문

- 항상 파일을 `open()` 함수로 열고, `close()` 함수로 닫아야 하는 일을 자동으로 처리
- with문을 이용하면 with 블록 내에서 파일을 열고 벗어나면 파일을 닫음

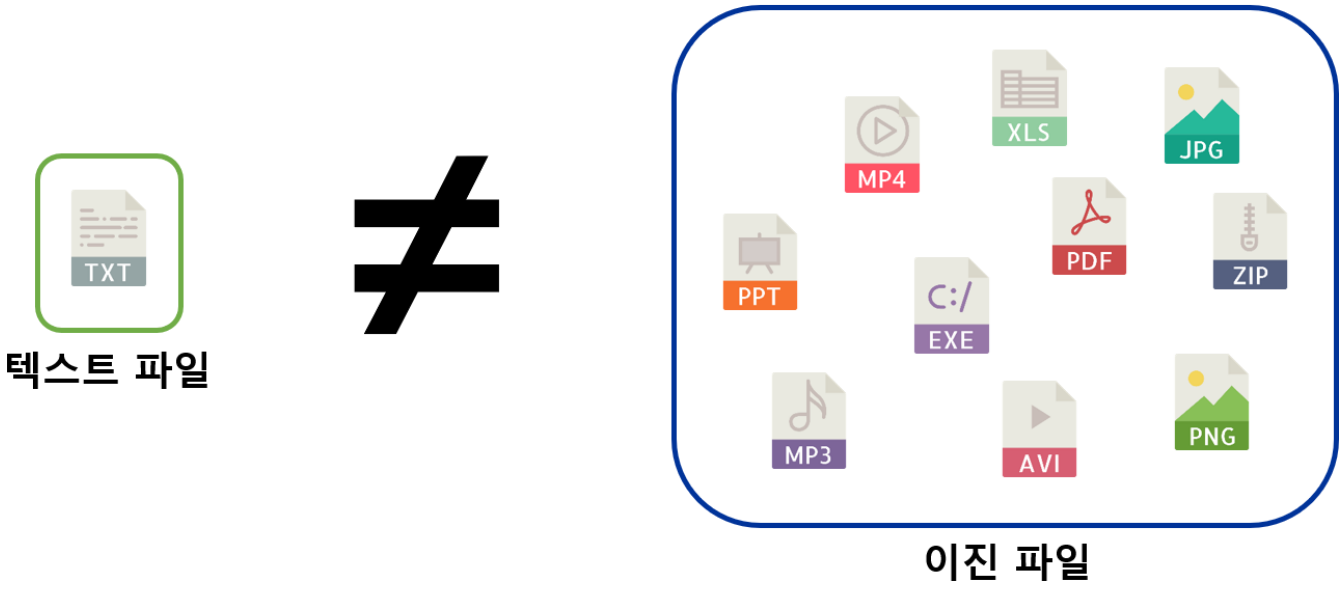
```
f = open("file.txt", 'w')
```

```
f.write("Hello Python")
f.close()
```

```
with open("file.txt", 'w') as f:
  f.write("Hello Python")
```

텍스트 파일 vs. 이진 파일

- 텍스트 파일(text file): 사람이 읽을 수 있는 텍스트로 구성된 파일
- 이진 파일(binary file): 텍스트가 아닌 비트 단위의 파일
- 이진 파일의 종류: 그림, 음악, 영상, 프로그램 파일 등



- 이진 파일(이미지) 예제

```
from google.colab import files
files.upload()
```


파일 선택 선택된 파일 없음 Upload widget is only available when the cell has been executed in the current browser session. Please rerun this cell to enable.
Saving unnamed.jpg to unnamed.jpg
{'unnamed.jpg': b'\xff\xd8\xff\xe0\x00\x10JFIF\x00\x01\x01\x00\x00\x01\x00\x01\x00\x00\xff\x


```
!ls
```

```
anthem.txt  dan.txt  file.txt  sample_data  unnamed.jpg
```

이진 파일 복사

- 하나의 이진 파일을 읽어서 다른 이진 파일로 쓰기 (복사)

```
input = open("unnamed.jpg", "rb")
output = open("unnamed_copy.jpg", "wb")
while True:
  fp = input.read(1)
  if not fp: break
  output.write(fp)

input.close()
output.close()
```

```
!ls
```

```
anthem.txt  dan.txt  file.txt  sample_data  unnamed_copy.jpg  unnamed.jpg
```

디렉토리 및 파일 처리

디렉토리 생성

- 디렉토리와 파일을 다루는 다양한 함수를 제공하는 shutil와 os 라이브러리
- `mkdir()`: 디렉토리 생성

```
import os
import shutil
os.mkdir('test')
os.mkdir('test/1')
os.mkdir('test/2')
os.mkdir('test/3')
```

```
!ls
```

```
anthem.txt  dan.txt  file.txt  sample_data  test  unnamed_copy.jpg  unnamed.jpg
```

```
!ls test
```

```
1  2  3
```

디렉토리 및 파일 복사

- `shutil.copy()`: 파일 복사
- `shutil.copytree()`: 디렉토리 전체 복사

```
import shutil
shutil.copy('unnamed.jpg', 'unnamed_copy2.jpg')
```

```
'unnamed_copy2.jpg'
```

```
!ls
```

```
anthem.txt  file.txt     test                unnamed_copy.jpg
dan.txt     sample_data  unnamed_copy2.jpg   unnamed.jpg
```

```
import shutil
shutil.copytree('test', 'test2')
```

```
'test2'
```

```
!ls
```

```
anthem.txt  file.txt     test   unnamed_copy2.jpg  unnamed.jpg
dan.txt     sample_data  test2  unnamed_copy.jpg
```

```
!ls test2
```

```
1  2  3
```

디렉토리 및 파일 확인

- `isdir()`: 디렉토리 존재 확인
- `isfile()`: 파일 존재 확인
- `exists()`: 디렉토리/파일 존재 확인

```
import os.path
print(os.path.isdir('test'))
print(os.path.isdir('test/1'))
print(os.path.isdir('test/4'))
print(os.path.isfile('unnamed.jpg'))
print(os.path.isfile('unnamed_copy.jpg'))
print(os.path.isfile('unnamed_copy2.jpg'))
print(os.path.exists('test2'))
print(os.path.exists('test2/1'))
print(os.path.exists('test2/4'))
```

```
True
True
False
True
True
True
True
True
False
```

디렉토리 목록 보기

- `os.walk()`: 디렉토리 목록 보기

```
import os
for dir_name, dir_list, file_names in os.walk('./'):
  for file_name in file_names:
    print(os.path.join(dir_name, file_name))
```

```
./anthem.txt
./unnamed.jpg
./unnamed_copy.jpg
./dan.txt
./unnamed_copy2.jpg
./file.txt
./.config/config_sentinel
./.config/.last_update_check.json
./.config/.last_survey_prompt.yaml
./.config/gce
./.config/.last_opt_in_prompt.yaml
./.config/active_config
./.config/.metricsUUID
./.config/logs/2020.06.17/16.18.24.878976.log
./.config/logs/2020.06.17/16.18.44.263522.log
./.config/logs/2020.06.17/16.18.30.389925.log
./.config/logs/2020.06.17/16.18.11.544396.log
./.config/logs/2020.06.17/16.17.52.116219.log
./.config/logs/2020.06.17/16.18.44.921040.log
./.config/configurations/config_default
./sample_data/README.md
./sample_data/anscombe.json
./sample_data/mnist_test.csv
./sample_data/california_housing_test.csv
./sample_data/mnist_train_small.csv
./sample_data/california_housing_train.csv
```

디렉토리 및 파일 삭제

- `os.remove()`: 파일 삭제
- `shutil.rmtree()`: 디렉토리 안에 모든 파일/디렉토리 삭제

```
import os
os.remove('unnamed_copy2.jpg')
os.remove('unnamed_copy.jpg')
```

```
!ls
```

```
anthem.txt  dan.txt  file.txt  sample_data  test  test2  unnamed.jpg
```

```
import shutil
shutil.rmtree('test')
shutil.rmtree('test2')
```

```
!ls
```

```
anthem.txt  dan.txt  file.txt  sample_data  unnamed.jpg
```

파일 크기

- `os.path.getsize()`: 파일의 크기를 바이트 단위로 출력

```
import os.path
print(os.path.getsize('unnamed.jpg'))
```

```
40229
```

파일 압축

- zipfile: 압축 관련 라이브러리
- `ZipFile()`: 압축 파일 열기
- `write()`: 압축 파일에 쓰기
- `extractall()`: 전체 압축 해제
- `close()`: 압축 파일 닫기

```
import zipfile
comp = zipfile.ZipFile('new.zip', 'w')
comp.write('unnamed.jpg')
comp.close()
```

```
!ls
```

```
anthem.txt  dan.txt  file.txt  new.zip  sample_data  unnamed.jpg
```

```
decomp = zipfile.ZipFile('new.zip', 'r')
decomp.extractall('new')
decomp.close()
```

```
!ls new
```

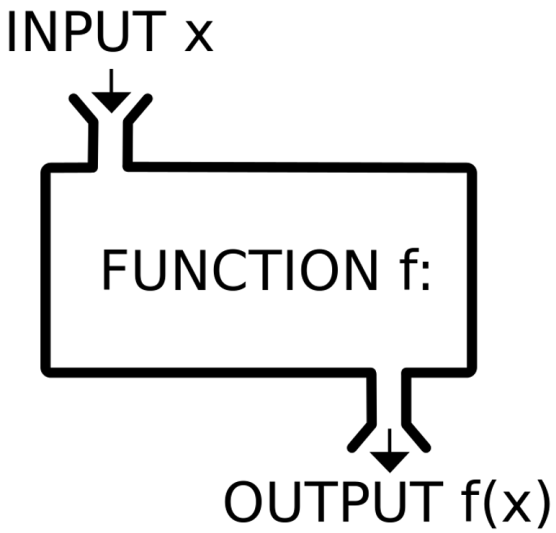
```
unnamed.jpg
```

함수(Function)

함수 기본

함수 개념

- 특정 값 X를 인자로 받고, 결과값을 반환: $Y = f(X)$
- 함수가 필요할 때마다 호출 가능
- 논리적인 단위로 분할 가능
- 코드의 캡슐화(Capsulation)
- 중복되는 소스코드를 최소화
- 소스코드의 재사용성을 높임



함수 선언

- 함수 선언 문법

```
def 함수명(매개변수):
  <수행문>
  return <반환값>
```

- def: 정의(definition)의 줄임으로 사용
- 함수명: 사용자가 임의로 지정
- 함수명 컨벤션(convention)
  - 짧고 명료한 이름
  - 소문자로 입력
  - 띄어쓰기는 '_' 기호 사용 (hello_world)
  - 동사와 명사를 함께 사용 (find_name)
- 매개변수(parameter): 함수에서 입력값으로 사용하는 변수
- 반환값: 함수에서 반환할 결과값 지정

매개변수와 반환값이 없는 함수

- 함수에 매개변수와 반환값이 없이 사용 가능
- 함수를 호출하면 함수의 수행문이 실행

```
def hello():
  print("Hello Python")

hello()
```

```
Hello Python
```

매개변수만 있는 함수

- 문자열 매개변수를 사용한 함수

```
def hello(string):
  print("Hello", string)

hello("Python")
```

```
Hello Python
```

반환 값만 있는 함수

- 문자열 반환 값을 사용한 함수

```
def hello():
  return "Hello Python"

hello()
```

```
'Hello Python'
```

매개변수와 반환 값이 있는 함수

- 정수형 매개변수와 반환 값을 사용한 함수

```
def square(num):
  return num * num

square(5)
```

```
25
```

매개변수가 여러개 있는 함수

- 정수형 매개변수 여러개를 사용한 함수
- 매개변수를 지정하여 호출 가능

```
def add(n1, n2):
  return n1 + n2

print(add(5, 8))
print(add(n2 = 5, n1 = 8))
```

```
13
13
```

키워드 매개변수

- 함수의 매개변수를 변수명을 지정하여 호출 가능

```
def add(n1, n2):
  return n1 + n2

print(add(n2 = 5, n1 = 8))
```

```
13
```

가변 매개변수

- 매개변수가 몇 개인지 알 수 없을 때 사용
- 매개변수 앞에 '*'을 표시

```
def sum(*args):
  result = 0
  for i in args:
    result = result + i
  return result

print(sum(1, 2, 3))
print(sum(1, 2, 3, 4, 5))
```

```
6
15
```

가변 키워드 매개변수

- 매개변수의 이름을 따로 지정하지 않고 사용
- 매개변수 앞에 '**'을 표시

```
def print_kwargs(**kwargs):
  print(kwargs)

print_kwargs(n1 = 5, n2 = 8)
print_kwargs(id = "Suan", pw = "1234")
```

```
{'n1': 5, 'n2': 8}
{'id': 'Suan', 'pw': '1234'}
```

초기값 매개변수

- 매개변수에 초기값을 설정하여 사용
- 함수에 매개변수를 사용하지 않을 때 초기값을 사용

```
def power(b = 2, n = 2):
  return pow(b, n)

print(power())
print(power(3))
print(power(5, 2))
print(power(n = 3))
```

```
4
9
25
8
```

여러 반환 값이 있는 함수

- 함수의 반환값은 하나
- 여러 반환값을 사용할 경우 튜플 형태로 반환

```
def plus_and_minus(n1, n2):
  return n1 + n2, n1 - n2

result = plus_and_minus(8, 5)
print(result)

result1, result2 = plus_and_minus(8, 5)
print(result1, result2)
```

```
(13, 3)
13 3
```

변수의 유효범위

유효 범위

- 변수는 유효한 범위가 존재

- 함수 안에서 선언된 변수는 함수 내부에서 유효함

```
def var_scope(a):
  a = a + 1

a = 10
var_scope(a)
print(a)
```

```
10
```

변수의 종류

- 지역 변수: 한정된 지역에서만 사용되는 변수
- 전역 변수: 프로그램 전체에서 사용되는 변수

```
a = 10
def func1():
  a = 20
  print(a)

def func2():
  print(a)

func1()
func2()
```

```
20
10
```

전역 변수 사용 global

- 함수 내부에서 전역 변수를 사용하기 위한 `global` 키워드

```
a = 10
def func1():
  global a
  a = 20
  print(a)

def func2():
  print(a)

func1()
func2()
```

```
20
20
```

모듈과 패키지

모듈(Module)

- 함수, 변수 그리고 클래스의 집합
- 다른 파이썬 프로그램에서 가져와 사용할 수 있는 파이썬 파일
- 파이썬에는 다른 사람들이 만들어 놓은 모듈이 굉장히 많음
- 사용자가 모듈은 직접 만들어서 사용할 수도 있음

모듈의 장점

- 단순성(Simplicity)
  - 전체 문제에 초점을 맞추기보다는 문제의 상대적으로 작은 부분에만 초점을 맞춤
  - 단일 모듈로 작업할 수 있는 작은 도메인
  - 개발이 쉬우며 오류 발생이 적음
- 유지보수성(Maintainability)
  - 일반적으로 모듈은 서로 다른 문제 영역간에 논리적 경계를 설정하도록 설계
  - 상호 의존성을 최소화하는 방식으로 모듈을 작성하여 단일 모듈을 수정하면 프로그램의 다른 부분에 영향을 미칠 가능성이 줄어듬
  - 모듈 외부의 응용 프로그램에 대해 전혀 알지 못해도 모듈을 변경할 수 있음
  - 개발팀이 대규모 응용 프로그램에서 공동으로 작업 할 수 있음
- 재사용성(Reusability)
  - 단일 모듈에서 정의된 기능은 응용 프로그램의 다른 부분에서 (적절히 정의된 인터페이스를 통해) 쉽게 재사용 가능
  - 중복 코드를 만들 필요가 없음
- 범위 지정(Scoping)
  - 일반적으로 모듈은 프로그램의 여러 영역에서 식별자 간의 충돌을 피하는 데 도움이 되는 별도의 네임 스페이스를 정의

모듈의 종류

- 사용자 정의 모듈: 사용자가 직접 정의해서 사용하는 모듈
- 표준 모듈: 파이썬에서 기본 제공하는 모듈
- 서드 파티 모듈: 외부에서 제공하는 모듈

- 파이썬 표준 모듈에 모든 기능이 있지 않음
- 서드 파티 모듈을 이용해 고급 프로그래밍 가능
- 게임 개발을 위한 pygame, 데이터베이스 기능의 SQLAlchemy, 데이터 분석 기능의 NumPy



파이썬 표준 모듈

- 파이썬에서 기본으로 내장된 유용한 속성과 함수들이 많음

```
import sys
print(sys.builtin_module_names)
```

```
('_ast', '_bisect', '_blake2', '_codecs', '_collections', '_datetime', '_elementtree', '_fun
```

```
print(dir(__builtins__))
```

```
['ArithmeticError', 'AssertionError', 'AttributeError', 'BaseException', 'BlockingIOError',
```

시간 모듈(datetime)

- 운영체제가 제공하는 시간 기능을 파이썬에서 사용할 수 있도록 만들어진 모듈
- 시간 모듈을 사용하기 위해서는 `import time` 필요

```
import time
print(dir(time))
```

```
['CLOCK_MONOTONIC', 'CLOCK_MONOTONIC_RAW', 'CLOCK_PROCESS_CPUTIME_ID', 'CLOCK_REALTIME', 'CL
```

- 시간 모듈 `time` 예제

```
import time
print(time)
print(time.time())
print(time.time())
print(time.time())

now = time.gmtime(time.time())
print(now)

year = str(now.tm_year)
```

```
month = str(now.tm_mon)
day = str(now.tm_mday)
print(year + "년", month + "월", day + "일")

hour = str(now.tm_hour)
minute = str(now.tm_min)
sec = str(now.tm_sec)
print(hour + "시", minute + "분", sec + "초")
```

```
<module 'time' (built-in)>
1593203327.0664105
1593203327.0666735
1593203327.066869
time.struct_time(tm_year=2020, tm_mon=6, tm_mday=26, tm_hour=20, tm_min=28, tm_sec=47, tm_wd
2020년 6월 26일
20시 28분 47초
```

- 날짜시간 모듈 `datetime`의 `date` 클래스 예제

```
from datetime import date
print(date)
print(date(2000, 1, 1))
print(date(year = 2010, month = 1, day = 1))
print(date.today())

today = date.today()
year = str(today.year)
month = str(today.month)
day = str(today.day)
weekday = "월화수목금토일"[today.weekday()]
print(year + "년", month + "월", day + "일", weekday + "요일")
```

```
<class 'datetime.date'>
2000-01-01
2010-01-01
2020-06-26
2020년 6월 26일 금요일
```

- 날짜시간 모듈 `datetime`의 `time` 클래스 예제

```
from datetime import time
print(time)
print(time(12, 0))
print(time(14, 30))
print(time(16, 30, 45))
print(time(18, 00, 15, 100000))

now = time(20, 40, 15, 20000)
hour = str(now.hour)
minute = str(now.minute)
sec = str(now.second)
```

```
msec = str(now.microsecond)
print(hour + "시", minute + "분", sec + "초", msec + "마이크로초")
```

```
<class 'datetime.time'>
12:00:00
14:30:00
16:30:45
18:00:15.100000
20시 40분 15초 20000마이크로초
```

- 날짜시간 모듈 `datetime`의 `datetime` 클래스 예제
- 날짜시간을 문자열로 표현하기 위한 `strftime()` 메소드 예제

표현	설명
%Y	년(YYYY)
%y	년(yy)
%m	월(mm)
%d	일(dd)
%A	요일
%H	시(24)
%I	시(12)
%p	AM, PM
%M	분(MM)
%S	초(SS)
%f	마이크로초

```
from datetime import datetime
print(datetime)
print(datetime(2020, 1, 1))
print(datetime(2020, 1, 1, 1, 15, 45))
print(datetime.now())
now = datetime.now()
print(now.strftime('%Y년 %m월 %d일 %H시 %M분 %S초'))
print(now.strftime('%y/%m/%d %p %I:%M:%S:%f'))
```

```
<class 'datetime.datetime'>
2020-01-01 00:00:00
2020-01-01 01:15:45
2020-06-26 20:32:46.271818
2020년 06월 26일 20시 32분 46초
20/06/26 PM  8:32:46:273260
```

수학 모듈(math)

- 파이썬에서 수학에 필요한 math 모듈 제공

```
import math
print(dir(math))
```

```
['__doc__', '__loader__', '__name__', '__package__', '__spec__', 'acos', 'acosh', 'asin', 'a
```

- `math` 모듈 대표 상수

상수	설명
`math.pi`	원주율
`math.e`	자연상수
`math.inf`	무한대

- `math` 모듈에서 제공하는 대표 함수

함수	설명
`math.factorial(x)`	x 팩토리얼
`math.gcd(a, b)`	a와 b의 최대공약수
`math.floor(x)`	x의 내림값
`math.ceil(x)`	x의 올림값
`math.pow(x, y)`	x의 y승
`math.sqrt(x)`	x의 제곱근
`math.log(x, base)`	base를 밑으로 하는 x 로그
`math.sin(x)`	x 라디안의 사인
`math.cos(x)`	x 라디안의 코사인
`math.tan(x)`	x 라디안의 탄젠트
`math.degrees(x)`	x 라디안을 도 단위로 변환
`math.radians(x)`	x 도를 라디안 단위로 변환

```
import math
print(math.factorial(3))
print(math.gcd(12, 24))
print(math.floor(math.pi))
print(math.ceil(math.pi))
print(math.pow(2, 10))
print(math.sqrt(10))
print(math.log(10, 2))
print(math.degrees(math.pi))
print(math.radians(180))
print(math.sin(math.radians(90)))
print(math.cos(math.radians(180)))
```

```
6
12
3
4
1024.0
3.1622776601683795
3.3219280948873626
180.0
3.141592653589793
1.0
-1.0
```

랜덤 모듈(random)

- 랜덤 모듈을 사용하기 위해서는 `import random` 필요
    - `random.random()`: 0.0~1.0 미만의 실수값 반환
    - `random.randint(1, 10)`: 1~10 사이의 정수 반환
    - `random.randrange(0, 10, 2)`: 0~10 미만의 2의 배수만 반환
    - `random.choice()`: 자료형 변수에서 임의의 값 반환
    - `random.sample()`: 자료형 변수에서 필요한 개수만큼 반환
    - `random.shuffle()`: 자료형 변수 내용을 랜덤으로 셔플

```
import random
print(random.random())
print(random.randint(1, 10))
print(random.randrange(0, 10, 2))
```

```
0.037180254395209 11
10
4
```

```
li = [10, 20, 30, 40, 50]
print(li)
print(random.choice(li))
print(random.sample(li, 2))
random.shuffle(li)
print(li)
```

```
[10, 20, 30, 40, 50]
30
[50, 30]
[20, 50, 30, 40, 10]
```

네임스페이스(Namespace)

- 모듈 호출의 범위 지정
- 모듈 이름에 alias를 생성하여 모듈의 이름을 바꿔 사용

```
import random as rd

print(rd.random())
print(rd.randrange(0, 10, 2))
```

```
0.6960806655231452
2
```

- from 구문을 사용하여 모듈에서 특정 함수 또는 클래스만 호출

```
from random import random, randrange
```

```
print(random())
print(randrange(0, 10, 2))
```

```
0.07750405361556745
4
```

- '*'을 사용하여 모듈 안에 모든 함수, 클래스, 변수를 가져옴

```
from random import *

print(random())
print(randrange(0, 10, 2))
```

```
0.4802513180943273
8
```