argparse

: 파이썬 인자값을 추가하는 모듈

```
import argparse

# 파서 만들기
parser = argparse.ArgumentParser(description = '테스트')

# 인자 추가하기
parser.add_argument('--a', required=True , help='입력1')
parser.add_argument('--b', required=False, default='dev', help='입력2')

# 인자 파싱하기
args = parser.parse_args()

print(args.a)
print(args.b)
```

Ex)

```
C:\Users\jky07\OneDrive\문서\python\argparse> python argparse1.py

# 실행 결과
usage: argparse1.py [-h] --a A [--b B]
argparse1.py: error: the following arguments are required: --a
```

`add_argument()` 메서드

- `name or flags` : 옵션 문자열의 이름이나 리스트

```
parser.add_argument('--test', '-t')     # fullname과 약자를 동시에 지정할 수 있다.
```

인자 앞에 —, - 가 부터 있으면 선택형 인자(optional), 붙어 있지 않으면 위치형 인자(positional)이다.

위치형 인자는 필수적으로 입력해야 하는 인자이다.

- `action` : 인자와 값을 받을 때 코드가 해석하는 방식을 지정 (기본값 : store)
- `const` : 일부 action 및 nargs 를 선택할 때 필요한 상숫값

`store` : default값이다. 인자 이름+스페이스바 후에 나오는 값을 인자 값으로 저장한다. (= 파싱)

```
# 코드
parser.add_argument('--a', action='store')

# cmd창
# > python argparse1.py --a 5

# 실행 결과
# 5

# parse_args() method 내에서 '--a 1'.split()이 일어나며 --a에 1이 저장된다.
# 이것이 파싱이다.
```

`store_const` : const 키워드 인자에 저장된 값을 저장한다. const=을 반드시 입력해야 한다.

```
# 코드
parser.add_argument('--a', action='store_const', const='10')

# cmd창
# > python argparse1.py --a

# 실행 결과
# 10
```

`store_true, store_false` : True, False 값을 저장하는 store_const의 일종이다. const를 쓰지 않아도 True, False 값이 저장된다.

```
#코드
parser.add_argument('--a1', action='store_true')
parser.add_argument('--a2', action='store_false')
parser.add_argument('--a3', action='store_true')

args = parser.parse_args()

print('args.a1 : ', args.a1)
print('args.a2 : ', args.a2)
print('args.a3 : ', args.a3)

#cmd창
# > python argparse1.py --a1 --a3

# 실행 결과
# args.a1 :  True
# args.a2 :  True
# args.a3 :  True
```

`append` : 리스트를 저장하고 인자가 불릴 때마다 리스트에 값을 추가한다.

```
# 코드
parser.add_argument('--a', action='append')

#cmd창
# > python agrparse1.py --a 3 --a 2 --a 1

# 실행 결과
# ['3','2','1']
```

`append_const` : 리스트를 저장하고 const 키워드 인자로 지정된 값을 리스트에 추가한다.

```
# 코드
parser.add_argument('--a', action='append_const', const='1,2,3')

#cmd창
# > python argparse1.py --a

# 실행 결과
# [1,2,3]
```

`count` : 키워드 인자가 등장한 횟수를 계산한다.

```
# 코드
parser.add_argument('--a', action='count')

# cmd창
# > python argparse1.py --a --a --a

# 실행 결과
# 3
```

- nargs : 인자의 값 개수 지정

보통 인자 하나당 하나의 값이 들어오지만 nargs는 한 인자에 여러 개의 값이 들어올 수 있게 한다.

```
# 코드
parser.add_argument('--a1', nargs=2)
parser.add_argument('--a2', nargs='?')
parser.add_argument('--a3', nargs='*')
parser.add_argument('--a4', nargs='+')

args = parser.parse_args()
```

```
print('N : ', args.a1)
print('? : ', args.a2)
print('* : ', args.a3)
print('+ : ', args.a4)
print('argparse.REMINDER : ', args.a5)

# cmd창
# > python argparse1.py --a1 1 2
# > python argparse1.py --a2 1
# > python argparse1.py --a3
# > python argparse1.py --a4 1

# 실행 결과
# N : ['1','2']
# ? : 1
# * : ['1']
```

`N` : N개의 값을 읽어들인다.

`?` : 0개 또는 1개의 값을 읽어들인다.

- 인자와 값을 모두 적은 경우 해당 값이 저장된다.
- 인자만 적은 경우 const 값이 저장된다.
- 아무것도 적지 않았으면 default 값이 저장된다.

`*` : 0개 이상의 값을 전부 읽어들인다.

`+` : 1개 이상의 값을 전부 읽어들인다.

- `default` : 인자가 명령행에 없을 경우 기본적으로 들어가는 값 지정

```
# 코드
parser.add_argument('--a', default=1)

# cmd창
# > python argparse1.py --a

# 실행 결과                          # -- default 값을 지정하지 않으면 None이 출력된다.
# 1
```

- `type` : 인자가 int, float, str, bool 등 어떤 type인지 지정

```
# 코드
parser.add_argument('--a',type=int)

# cmd창
# > python argparse1.py --a 1
# > python argparse1.py --a 안녕   # -- error 발생
```

```
# 실행 결과
# 1
```

- choices : 인자 값의 범위 지정

```
# 코드
parser.add_argument('--a', choices=['ex1', 'ex2', 'ex3'])

# cmd창
# > python argparse1.py --a ex1
# > python argparse1.py --a test   # -- error 발생

# 실행 결과
# ex1
```

정해진 값 중 하나를 선택해야 할 때 사용한다.

- required : 필수 입력 지정

```
# 코드
parser.add_argument('--a', required=True)

# cmd창
# > python argparse1.py --a 1
# > python argparse1.py --a      # -- error 발생

# 실행 결과
# 1
```

required 옵션에 True를 주게 되면, 인자값이 전달되지 않을 시 실행이 되지 않는다.

- help : 인자가 하는 일에 대한 간단한 설명

```
# 코드
parser.add_argument('--a', help='입력')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: argparse1.py [-h] [--a A]

# 테스트

#optional arguments:
```

```
#  -h, --help  show this help message and exit
#  --a A       입력
```

인자에 대한 설명을 쓴다.

command 창에서 —help 또는 -h를 입력할 경우 각 인자에 대한 help가 출력된다.

- `matavar` : 사용자 메시지에 사용되는 인자의 이름

```
# 코드
parser.add_argument('--a', metavar='인자의 이름')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: argparse1.py [-h] [--a 인자의 이름]

# 테스트

# optional arguments:
#  -h, --help  show this help message and exit
#  --a 인자의 이름
```

- `dest` : parse_args()가 반환하는 객체에 추가될 어트리뷰트의 이름

```
# 코드
parser.add_argument('--a', dest='어트리뷰트')
```

`ArgumentParser()` 메서드

- `prog` : 프로그램 이름 (기본값 : sys, argv[0])

```
# 코드
parser = argparse.ArgumentParser(prog='myprogram')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: myprogram [-h] --a A

# optional arguments:
#  -h, --help  show this help message and exit
#  --a A       입력1
```

참고

help에서 %(prog)s를 활용하여 prog를 사용할 수 있습니다.

```
# 코드
parser = argparse.ArgumentParser(prog='myprogram')
parser.add_argument('--a', required=True , help=' %(prog)s 입니다.')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: myprogram [-h] --a A

# optional arguments:
#  -h, --help  show this help message and exit
#  --a A       myprogram 입니다.
```

- `usage` : 프로그램 사용법을 설명하는 문자열

```
# 코드
parser = argparse.ArgumentParser(prog='PROG', usage='%(prog)s [options]')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: PROG [options]

# optional arguments:
#  -h, --help  show this help message and exit
#  --a A       입력1
```

- `description` : 인자 도움말 전에 표시할 텍스트 (기본값 : None)

```
# 코드
parser = argparse.ArgumentParser(description='test description')

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: argparse1.py [-h] --a A

# test description

# optional arguments:
#  -h, --help  show this help message and exit
#  --a A       입력1
```

- epilog : 인자 도움말 후에 표시할 텍스트 (기본값 : None)

```
# 코드
parser = argparse.ArgumentParser(epilog="test epilog")

# cmd창
# > python argparse1.py -h

# 실행 결과
# usage: [argparse1.py](http://argparse1.py/) [-h] --a A

# optional arguments:
#  -h, --help  show this help message and exit
#  --a A       입력1

# test epilog
```

- parents : ArgumentParser 객체들의 리스트

- formatter_class : 도움말 출력을 사용자 정의하기 위한 클래스
  - argparse.RawDescriptionHelpFormatter : decription과 epilog가 이미 올바르게 포맷되어 있어 줄 바꿈 되어서는 안 된다는 것을 카리킨다.
  - argparse.RawTextHelpFormatter : 모든 종류의 도움말 텍스트에 있는 공백을 유지한다.
  - argparse.ArgumentDefaultsHelpFormatter : 기본값에 대한 정보를 각각의 인자 도움말 메시지에 자동으로 추가한다.
  - argparse.MetavarTypeHelpFormatter : 각 인자 값의 표시 이름으로 type 인자의 이름을 사용한다.

- prefix_chars : 선택 인자 앞에 붙는 문자 집합 (기본값 : '-')

—test / -t 와 같이 앞에 붙는 접두사를 다른 것으로 바꿀 수 있다.

- fromfile_prefix_chars : 추가 인자를 읽어야 하는 파일 앞에 붙는 문자 집합 (기본값 : None)

```
# 코드
with open('ex.txt','w', encoding='utf-8-sig') as fp:
    fp.write('-f\ntest')
    parser = argparse.ArgumentParser(fromfile_prefix_chars='@')
```

```
    parser.add_argument('-f')
    parser.parse_args(['-f', 'foo', '@ex.txt'])
```

[' -f ', ' foo ', ' @ex.txt ']는 [' -f ', ' foo ', ' -f ', ' test '] 와 동등하게 취급됩니다.

- `argument_default` : 인자의 전역 기본값 (기본값 : None)

- `add_help` : 파서에 -h/—help 옵션을 추가 (기본값 : True)

같은 옵션의 문자열을 가진 두 개의 액션을 허용하지 않는다.

- `allow_abbrev` : 약어가 모호하지 않으면 긴 옵션을 축약할 수 있도록 함 (기본값 : True)