Python Provided Module

날짜 및 시간 데이터

- 시간 표현
  - ✓ TimeStamp: 컴퓨터에서 일반적으로 시간을 표현하는 방법으로 1970년 1월 1일 자정을 기준으로 해서 초 단위나 밀리초 단위로 측정한 절대시간
  - ✓ UTC(Universal Time Coordinated: 협정 세계시): 1972년부터 시행된 국제 표준시로 세슘 원자의 진동수에 의거한 초의 길이
  - ✓ GMT(Greenwich Mean Time: 그리니치 평균시): 런던 그리니치 천문대의 자오선상에서의 평균 태양시로 1972년 부터는 UTC를 사용하지만 동일한 의미로 사용
  - ✓ LST(Local Standard Time: 지방 표준시): UTC를 기준으로 경도 15도마다 1시간 차이가 발생하는 시간으로 대한민국은 동경 135도를 기준으로 UTC 보다 9시간 빠름
  - ✓ DTS(Daylight Saving Time: 일광절약 시간제): 에너지 절약을 목적으로 시간을 한 시간씩 앞당기거나 뒤로 미루는 제도

날짜 및 시간 데이터

- struct_time
  - ✓ 컴퓨터는 TimeStamp 형식으로 시간을 관리하는데 TimeStamp는 정수 형태로 만들어져 있어서 인간이 알아보기 힘들기 때문에 인간이 알아보기 쉽도록 년, 월, 일, 시, 분, 초 형태로 분할해서 시간을 저장하는 시퀀스 인스턴스
  - ✓ 속성
    - tm_year: 년도
    - tm_mon: 월(1-12)
    - tm_mday: 월에서의 일(1-31)
    - tm_hour: 시(0-23)
    - tm_min: 분(0-59)
    - tm_sec: 초(0-61)
    - tm_wday: 요일을 숫자로 표현하는데 월요일이 0
    - tm_yday: 1월 1일부터 지나온 누적된 날짜
    - tm_isdst: 서머타임을 나타내는 숫자로 0, -1, 1

날짜 및 시간 데이터

- time 모듈
  - ✓ 매개변수를 전달하지 않으면 현재 시스템의 타임스탬프 값을 이용
    - time.time(): 1970년 1월 1일 자정부터 지나온 시간을 초 단위의 float 값으로 리턴
    - time.gmtime(): UTC 기준의 struct_time 타입으로 리턴
    - time.localtime(): localtime 기준의 struct_time 타입으로 리턴
  - ✓ time.sleep(sec): 현재 동작중인 프로세스를 sec 초 만큼 정지
  - ✓ time.asctime(t): struct_time 타입을 매개변수로 받아서 문자열 형식의 시간으로 리턴
  - ✓ time.gmtime(t): 누적된 시간을 리턴

날짜 및 시간 데이터

- time 모듈

```
import time

#현재 시간을 실수로 가져오기
print(time.time())
#현재 시간을 struct_time 형식으로 가져오기
print(time.gmtime())
#년도만 가져오기
print(time.localtime().tm_year)
t1 = time.time()
#10초 대기
time.sleep(10)
t2 = time.time()
#시간에서 시간을 빼서 경과한 시간 만들기
spendtime = t2-t1
print("Wait seconds ", spendtime)
```

```
1629356292.424618
time.struct_time(tm_year=2021, tm_mon=8, tm_mday=19, tm_hour=6, tm_min=58, tm_sec=12,
tm_wday=3, tm_yday=231, tm_isdst=0)
2021
```

날짜 및 시간 데이터

- datetime 패키지
  - ✓ 하위 클래스
    - datetime 클래스 : 날짜+시간 복합 정보
    - date 클래스: 날짜 정보
    - time 클래스: 시간 정보
    - timedelta 클래스: 시간 차이 정보
    - tzinfo 클래스: 시간대 정보

날짜 및 시간 데이터

- datetime 패키지
  - ✓ date 클래스 생성
    - datetime.date(년, 월, 일)
    - date.today()
  - ✓ time 클래스
    - datetime.time(시, 분, 초, 밀리초, tzinfo)

날짜 및 시간 데이터

- datetime 패키지
  - ✓ datetime.datetime 클래스
    - 날짜와 시간 정보를 가지는 클래스
    - 생성자: datetime(year, month, day[, hour[,minute[,second[,microsecond[,tzinfo]]]]])
    - 클래스 메서드
      - today(), now([tzinfo]): 현재 날짜 및 시각으로 생성
      - strptime(date_string, format): 문자열 -> datetime
      - fromtimestamp(timestamp): timestamp -> datetime
      - fromordinal(ordinal): 1년1월1일 부터의 누적된 날짜 -> datetime
      - combine(date, time): date + time -> datetime

날짜 및 시간 데이터

- ❖ datetime 패키지
  - ✓ datetime.datetime 클래스
    - 속성: 읽기 전용
      - year, month, day, hour, minute, second, microsecond
      - tzinfo
    - 인스턴스 메서드
      - weekday(): 요일 {0:월, 1:화, 2:수, 3:목, 4:금, 5:토, 6:일}
      - isoweekday(): 1부터 시작하는 숫자로 요일을 리턴
      - strftime(format): datetime -> format에 해당하는 문자열을 리턴
      - toordinal(): datetime -> proleptic Gregorian ordinal (엑셀 형식의 날짜)
      - date(): datetime -> date
      - time(): datetime -> time
    - 문자열 변환
      - strptime(date_string, format): 문자열 -> datetime
      - strftime(format): datetime -> 문자열

날짜 및 시간 데이터

- ❖ datetime 패키지
  - ✓ datetime.datetime 클래스
    - ● 포맷 기호
      - ◆ %Y: 네자리 년도
      - ◆ %y: 두자리 년도
      - ◆ %m: 두자리 월
      - ◆ %d: 두자리 일
      - ◆ %H: 24시간
      - ◆ %I: 12시간
      - ◆ %M: 분, zero-padded decimal
      - ◆ %S: 초, zero-padded decimal

날짜 및 시간 데이터

❖ datetime 패키지

```
import datetime
#현재 시간을 저장
dt = datetime.datetime.now()
#전체 시간을 출력
print("현재시간:", dt)
#각 속성을 분할해서 출력
print(dt.year, dt.month, dt.day, dt.hour, dt.minute, dt.second, dt.microsecond, dt.tzinfo)
print("요일:",dt.weekday())

#문자열을 가지고 datetime 생성
dt1 = datetime.datetime.strptime("2015-12-31 11:32", "%Y-%m-%d %H:%M")
print(dt1)

#datetime을 문자열로 생성
s = dt1.strftime('%Y %m %d %H %M %S')
print(s)
```

```
현재시간: 2021-08-19 16:03:30.857444
2021 8 19 16 3 30 857444 None
요일: 3
2015-12-31 11:32:00
2015 12 31 11 32 00
```

날짜 및 시간 데이터

- ❖ datetime 패키지
  - ✓ datetime, date, time 변환
    - ● datetime -> date 또는 time

      date() 메서드 또는 time() 메서드
    - ● date + time -> datetime

      combine() 메서드
  - ✓ timedelta와 날짜 시간 연산
    - ● 빼기

      datetime - datetime => timedelta
    - ● 더하기

      datetime + timedelta => datetime

날짜 및 시간 데이터

- datetime 패키지
  - ✓ timedelta 클래스
    - days: 일수
    - seconds: 초. (0 ~ 86399)
    - microseconds: 마이크로 초 (0 and 999999)
    - total_seconds(): 모든 속성을 초단위로 모아서 변환
  - ✓ dateutil 패키지
    - parse() 함수: 많이 사용되는 날짜 시간 관련 문자열 형식을 스스로 판단

날짜 및 시간 데이터

❖ datetime 패키지

```
import datetime

#현재 날짜 및 시간을 저장
dt = datetime.datetime.now()
#날짜와 시간을 추출
print(dt.date())
print(dt.time())

#날짜 및 시간을 가지고 datetime을 생성
d = datetime.date(2015, 12, 31)
print(d)
t = datetime.time(11, 31, 29)
print(datetime.datetime.combine(d, t))

#2개의 datetime의 차이
dt1 = datetime.datetime(2016, 2, 19, 14, 11, 9)
dt2 = datetime.datetime(2016, 1, 2, 13, 16, 23)
td = dt1 - dt2
print(td)
print(td.days, td.seconds, td.microseconds)

#문자열을 날짜로 변환
from dateutil.parser import parse
print(parse('2016-04-16'))
```

```
2021-01-25
21:38:04.549862
2015-12-31
2015-12-31 11:31:29
48 days, 0:54:46
48 3286 0
2016-04-16 00:00:00
```

수치 데이터 표현

- fractions 모듈의 Fraction 클래스를 이용한 분수 표현
  - ✓ fractions 모듈을 이용해서 분수 표현 가능
  - ✓ 인스턴스 생성
    - 5/7을 표현하고자 할 때 Fraction('5/7') 또는 Fraction(5,7)
    - 실수 문자열을 입력해서도 생성 가능
    - from_float(실수)
    - from_decimal(10진수)
  - ✓ 데이터의 numerator 와 denominator를 이용해서 분자와 분모를 따로 추출 가능
  - ✓ 메서드
    - __floor__(): 해당 유리수를 넘지 않는 최대의 정수 리턴(내림)
    - __ceil__(): 해당 유리수를 넘는 최소의 정수 리턴(올림)
    - __round__(): 반올림
    - fractions.gcd(a, b): 최대 공약수
    - 연산자 중복으로 산술 연산 가능

수치 데이터 표현

❖ 분수 표현

```
from fractions import Fraction
#분수를 생성해서 더라기
a = Fraction(5,7) + Fraction('2/5')
print(a)
#분수 빼기
a = a - Fraction(5,7)
print(a)
#분수 나누기
a = a * Fraction(5,7)
print(a)
#소수를 이용해서 분수 생성
a = Fraction("1.24")
print(a)
#버림, 올림, 반올림
print(a.__floor__())
print(a.__ceil__())
print(a.__round__())
```

```
39/35
2/5
2/7
31/25
1
2
1
```

파일 시스템

- os.path
  - ✓ 파일 경로를 생성 및 수정하고 파일 정보를 다룰 수 있게 해주는 모듈
  - ✓ os.path.abspath(path): 현재 경로를 prefix로 하여 입력받은 경로를 절대 경로로 바꿔서 리턴
  - ✓ os.path.basename(path): 입력받은 경로의 기본 이름(base name)을 리턴
  - ✓ os.path.commonprefix(path_list): 입력받은 path_list로부터 공통적인 prefix를 추출해서 리턴
  - ✓ os.path.dirname(path): 입력받은 파일/디렉터리의 경로를 리턴
  - ✓ os.path.exists(path): 입력받은 경로가 존재하면 True를 반환하고 존재하지 않는 경우는 False를 리턴하는데 리눅스와 같은 OS에서는 파일이나 디렉터리가 존재하지만 읽기 권한이 없는 경우에도 False를 리턴
  - ✓ os.path.expanduser(path): 입력받은 경로 안의 "~"를 현재 사용자 디렉터리의 절대 경로로 대체
  - ✓ os.path.join(path1[,path2[,...]]): 해당 OS 형식에 맞도록 입력 받은 경로를 연결(입력 중간에 절대 경로가 나오면 이전에 취합된 경로는 제거하고 다시 연결)
  - ✓ os.path.split(path): 입력 받은 경로를 디렉터리 부분과 파일 부분으로 나누어서 리턴하는데 단순한 문자열 연산이므로 실제 파일의 존재 여부는 확인하지 않음
  - ✓ os.path.splitext(path): 입력 받은 경로를 확장자 부분과 그 외의 부분으로 나눔

파일 시스템

- os.path
  - ✓ os.path.getatime(path): 입력받은 경로에 대한 최근 접근 시간을 리턴(리턴되는 값은 epoch - 1970년 1월 1일 이후 시간을 초단위로 리턴하는데 파일이 없거나 권한이 없는 경우 os.error 예외 발생)
  - ✓ os.path.getmtime(path): 입력받은 경로에 대한 최근 변경 시간을 리턴(파일이 없거나 권한이 없는 경우 os.error 예외 발생)
  - ✓ os.path.getctime(path): 입력받은 경로에 대한 생성 시간을 리턴(유닉스와 같은 운영체제에서는 생성 시간이 아닌 최근 변경 시간을 반환할 수 있으며 파일이 없거나 권한이 없는 경우 os.error 예외 발생)
  - ✓ os.path.getsize(path): 입력받은 경로에 대한 바이트 단위의 파일 크기를 리턴(파일이 없거나 권한이 없는 경우 os.error 예외 발생)
  - ✓ os.path.isfile(path): 경로가 파일인지 아닌지 검사해서 파일인 경우에는 True를 반환하고 그 외의 경우 False를 리턴
  - ✓ os.path.isdir(path): 경로가 디렉터리인지 아닌지 검사해서 디렉터리인 경우에는 True를 반환하고 그 외의 경우에는 False를 반환

파일 시스템

❖ 경로 추출

```
import os.path
#현재 경로에 temp를 붙여서 리턴
print(os.path.abspath('temp'))

#마지막 경로만 리턴
print(os.path.basename('c:₩₩data'))

#문자열 차원에서 추출하므로 이상한 결과가 리턴될 수 있음
print(os.path.commonprefix(['C:₩₩Python30₩₩Lib', 'C:₩₩Python30',
'C:₩₩Python30₩₩Tools']))
print(os.path.commonprefix(['C:₩₩Python26₩₩Lib', 'C:₩₩Python25₩₩Tools']))
```

```
/Users/adam/Documents/lecture/PYTHON/1.Basic/temp
c:₩data
C:₩Python30
C:₩Python2
```

파일 시스템

❖ 파일에 대한 정보 확인

```
import os
import time

#파일의 경로를 변경해서 수행
print("접근시간:",os.path.getatime('./8.PythonProvidedModule.py'))

#타입 변환
print(time.gmtime(os.path.getatime('./8.PythonProvidedModule.py')))
print("수정시간:",os.path.getmtime('./8.PythonProvidedModule.py'))
print("생성시간:",os.path.getctime('./8.PythonProvidedModule.py'))

print(os.path.split('./8.PythonProvidedModule.py'))
print(os.path.splitext('./8.PythonProvidedModule.py'))
```

```
접근시간: 1689199938.4436746
time.struct_time(tm_year=2023, tm_mon=7, tm_mday=12, tm_hour=22, tm_min=12,
tm_sec=18, tm_wday=2, tm_yday=193, tm_isdst=0)
수정시간: 1689199938.0319443
생성시간: 1689199938.0319443
('.', '8.PythonProvidedModule.py')
('./8.PythonProvidedModule', '.py')
```

파일 시스템

- glob 모듈
  - ✓ glob 모듈은 윈도우의 dir 명령어나 리눅스의 ls 명령어와 유사한 기능을 제공하는데 디렉토리 내의 파일 및 디렉토리를 순회하고자 할 때 사용
  - ✓ glob(path): path 안의 파일 및 디렉토리의 리스트를 리턴하는데 * 이나 ? 와 같은 와일드 카드 문자 사용 가능
  - ✓ iglob(path): 리스트 대신에 iterator를 리턴

파일 시스템

- glob

```
import os, glob
file_list = os.listdir("./")
print(file_list)
print(glob.glob('./*.py'))
print(glob.iglob('./*.py'))
```

```
['mymath.py', '7.ExceptionHandling.py', 'map.html', '5.oop.py', '4.Function.py',
'3.ControlStatement.py', '2.Variable_Operator.py', '8.PythonProvidedModule.py',
'graph.png', 'venv', '.git', '6.DataType.py', '.idea']
['./mymath.py', './7.ExceptionHandling.py', './5.oop.py', './4.Function.py',
'./3.ControlStatement.py', './2.Variable_Operator.py', './8.PythonProvidedModule.py',
'./6.DataType.py']
<generator object _iglob at 0x10eedb900>
```

운영체제 관련 모듈

- os 모듈
  - ✓ getcwd(): 현재 작업 디렉터리의 위치를 리턴
  - ✓ chdir(path): 현재 작업 디렉터리 위치를 변경
  - ✓ os.access(path, mode): path에 mode 작업 가능 여부 리턴
    - mode
      - F_OK: 존재 여부
      - R_OK: 읽기 가능 여부
      - W_OK: 쓰기 가능 여부
      - X_OK: 실행 가능 여부
  - ✓ listdir(path): 파일과 디렉토리 목록을 리스트로 리턴
  - ✓ mkdir(path[, mode]): 디렉토리 생성
  - ✓ makedirs(path[, mode]): 디렉토리를 재귀적으로 생성하는데 디렉토리가 존재하면 예외 발생

운영체제 관련 모듈

❖ 현재 작업 디렉토리 정보 추출

```
import os
print("현재 작업 디렉토리:",os.getcwd())
print("현재 작업 디렉토리에 대한 정보:", os.stat(os.getcwd()))
```

```
현재 작업 디렉토리: /Users/adam/Documents/lecture/PYTHON/1.Basic
현재 작업 디렉토리에 대한 정보: os.stat_result(st_mode=16895, st_ino=385921,
st_dev=16777225, st_nlink=43, st_uid=501, st_gid=20, st_size=1376, st_atime=1629362510,
st_mtime=1629362510, st_ctime=1629362510)
```

운영체제 관련 모듈

- os 모듈
  - ✓ name: 운영체제 이름을 가지고 있는 속성
  - ✓ environ: 환경변수 목록을 dict로 리턴
  - ✓ getenv(varname[, value]): 특정 환경 변수의 값 리턴
  - ✓ putenv(varname, value): 환경변수 설정
  - ✓ getpid(): 현재 프로세스 아이디 리턴
  - ✓ system(command): command 실행
  - ✓ startfile(path, [operation]): path에 해당하는 파일을 기본 프로그램을 이용해서 실행하는데 operation 자리에 직접 프로그램을 지정 가능
  - ✓ exec(path, arg): path에 해당하는 새로운 프로그램을 실행

운영체제 관련 모듈

❖ 환경변수 추출과 프로세스 실행

```
import os
print("운영체제:", os.name)
print("환경변수 목록:", os.environ)
print("현재 프로세스 id:", os.getpid())
os.system("ls") #windows 에서는 os.system('dir')
```

운영체제 관련 모듈

- sys 모듈
  - ✓ 파이썬 인터프리터와 관련된 정보와 기능을 제공하는 모듈
    - argv: 파이썬 파일이 실행될 때 넘어온 매개변수 정보
    - exec_info(): 예외가 발생했을 때 예외 정보를 튜플로 리턴
    - prefix, exec_prefix: 파이썬이 설치된 경로
    - executable: 파이썬 인터프리터의 실행파일 경로
    - exit(정수): 0으로 대입하면 정상 종료 그렇지 않은 경우는 비정상 종료
    - getrefcount(object): object의 referencecount 리턴
    - modules: 현재 로딩된 모듈의 이름을 dict로 리턴
    - path: 모듈을 참조하는 경로의 순서를 list로 리턴
    - getdefaultencoding(): 현재 사용 중인 기본 문자열 인코딩을 리턴

운영체제 관련 모듈

- sys 모듈

```
import sys
print("파이썬이 설치된 경로:", sys.prefix)
msg = "Hello"
print("msg의 참조 개수:", sys.getrefcount(msg))
s1 = msg
print("msg의 참조 개수:", sys.getrefcount(msg))
s2 = msg
print("msg의 참조 개수:", sys.getrefcount(msg))
print("현재 인코딩 방식:", sys.getdefaultencoding())
print("현재 사용 중인 모듈:", sys.modules)
```

copy

- 복사
  - ✓ python에서의 복사는 2가지가 있음
  - ✓ 하나는 인스턴스의 참조를 복사하는 것이고 다른 하나는 인스턴스 자체를 복사하는 것
  - ✓ 파이썬에서는 리터럴을 대입하는 경우 리터럴을 저장하고 그 참조를 복사

```
a = 1
b = 1
print(id(a))
print(id(b))
```

1674327824
1674327824

- 위의 문장은 1이라는 인스턴스를 저장하고 그 주소를 a라는 변수에 대입하고 동일한 인스턴스의 주소를 b에 저장했으므로 동일한 id가 출력

copy

- 복사
  - ✓ 변수를 다른 변수에 대입하면 기본적으로 데이터의 참조를 복사

```
a = [1,2,3]
b = a
a[0]=100

print("a의 id:", id(a));
print("b의 id:", id(b));

print("a의 첫번째 요소:",a[0])
print("b의 첫번째 요소:",b[0])
```

```
a의 id: 44029232
b의 id: 44029232
a의 첫번째 요소: 100
b의 첫번째 요소: 100
```

copy

- 얕은 복사
  - ✓ python은 copy 모듈의 copy와 deepcopy 메서드를 이용해서 얕은 복사와 깊은 복사를 지원
  - ✓ copy는 얕은 복사를 지원하는 메서드
  - ✓ 얕은 복사를 하게 되면 인스턴스를 복제해서 다른 곳에 복사 한 후 대입
  - ✓ 얕은 복사는 재귀적으로 인스턴스를 복제해서 복사 작업을 수행해주지는 않d,a
  - ✓ 한번의 복사만 수행
  - ✓ list의 슬라이싱과 dict 의 copy 메서드도 얕은 복사

```
a = [1,2,3]
b = a[0:3]

a[0]=300
print("a[0]:", a[0])
print("b[0]:", b[0])
```

```
a[0]: 300
b[0]: 1
```

copy

- weak copy

```
import copy
a = [1,2,3]
x = [a, 100]
y = copy.copy(x)

#copy를 했으므로 2개의 아이디는 다름
print("x의 아이디:", id(x));
print("y의 아이디:", id(y));

#두번째 데이터는 일반적인 데이터이므로 복제가 이루어짐
x[1] = 200
print("x[1]:",x[1])
print("y[1]:",y[1])

#첫번째 데이터는 다시 데이터의 모임이므로 주소의 복제를 수행
x[0][0] = 300
print("x[0][0]:",x[0][0])
print("y[0][0]:",y[0][0])
```

```
x의 아이디: 2491496
y의 아이디: 46087544
x[1]: 200
y[1]: 100
x[0][0]: 300
y[0][0]: 300
```

copy

- 깊은 복사
  - ✓ deepcopy는 깊은 복사를 지원하는 메서드
  - ✓ 깊은 복사를 하게 되면 인스턴스를 복제해서 다른 곳에 복사 한 후 대입
  - ✓ 깊은 복사는 재귀적으로 인스턴스를 복제해서 복사 작업을 수행

```
import copy
a = [1,2,3]
x = [a, 100]
y = copy.deepcopy(x)

#copy를 했으므로 2개의 아이디는 다름
print("x의 아이디:", id(x));
print("y의 아이디:", id(y));

#첫번째 데이터는 다시 데이터의 모임이므로 재귀적으로 복제를 수행
x[0][0] = 300
print("x[0][0]:",x[0][0])
print("y[0][0]:",y[0][0])
```

```
x의 아이디: 140408537225152
y의 아이디: 140408537232128
x[0][0]: 300
y[0][0]: 1
```

메모리 정리

- 약한 참조
  - ✓ 클래스 인스턴스의 속성이 만약 다른 클래스의 인스턴스를 참조하거나 동일 클래스의 다른 인스턴스를 참조할 가능성이 높다면 이는 메모리 누수가 발생할 가능성이 매우 높은 지점이 됨
  - ✓ 클라이언트 프로그램의 생애주기는 대부분 짧기 때문에 문제가 되지 않을 수 있지만 서버와 같이 생애 주기가 길거나, 매우 많은 인스턴스를 생성하고 연결하는 동작을 하는 경우에 메모리 누수가 큰 문제가 될 수 있음
  - ✓ 파이썬에서는 이러한 문제를 조금 간단히 처리하기 위해서 약한 참조를 제공
  - ✓ 약한 참조는 말 그대로 대상 인스턴스를 참조는 하지만 대상 인스턴스에 대한 소유권을 주장하지 않는 즉 reference count를 올리지 않는 참조를 의미
  - ✓ 약한 참조는 weakref 모듈의 ref 라는 클래스를 통해서 생성 가능
  - ✓ 특정 대상에 대해 약한 참조를 만들 때는 weakref.ref(target) 으로 ref()에 인자로 넘겨 약한 참조 인스턴스를 생성
  - ✓ 생성된 약한 참조로부터 참조 대상을 얻으려 할 때는 약한 참조 인스턴스를 호출
  - ✓ 참조하려는 대상이 파괴되었다면 약한 참조는 참조 대상에 대한 액세스를 요청받을 때 None 을 리턴

메모리 정리

❖ 약한 참조

```
import weakref

class Temp:
  def __init__(self):
    self.value = 1
  def __del__(self):
    print('인스턴스가 파괴됩니다.')

a = Temp()
b = weakref.ref(a)
a = 1
```

인스턴스가 파괴됩니다.

Thread

- Thread
  - ✓ Task: 일 또는 작업이라는 말로 번역하며 프로세스와 Thread를 포함한 개념
  - ✓ Process: 운영체제로부터 자원을 할당받아 동작하는 독립된 프로그램
  - ✓ Thread: 프로세스 내의 명령어 블록으로 시작점과 종료점을 가지는 일련된 하나의 작업 흐름으로 혼자서 존재할 수 없고 항상 Process 내에 속해서 존재
  - ✓ 순차적으로 동작하는 문장들의 단일 집합으로 다른 Thread와 자원을 공유할 수 있으며 실행 중에 멈출 수 있으며(제어권을 다른 Thread에게 넘김) 동시에 서로 다른 Thread를 수행하는 것이 가능
  - ✓ 프로세스는 제어권을 가지면 종료 될 때까지 제어권의 이동이 불가능하고 다른 프로세스와 자원을 공유하지 않으며 통신을 이용해서만 다른 프로세스와 자원을 공유할 수 있음
  - ✓ 하나의 프로세스 안에 있는 스레드들은 각각 독립적으로 스택을 가지고 실행되지만 코드와 데이터는 공유
  - ✓ 스레드의 실행은 독립적이어서 다른 스레드 와의 실행 순서에 관한 어떠한 가정도 할 수 없음

Thread

- Thread
  - ✓ Python은 하나의 스레드는 하나의 코어에서만 동작시킬 수 있음
  - ✓ Python에서 스레드를 생성할 때는 threading 모듈의 Thread 클래스를 이용
  - ✓ 만드는 방법
    - Thread라는 생성자를 이용해서 생성: target에 스레드로 동작할 함수를 지정하고 args에 함수에 전달할 매개변수로 튜플을 전달하면 됨
    - Thread 클래스를 상속받아서 run 메서드를 재정의해도 됨
  - ✓ 스레드의 시작은 스레드 인스턴스가 start() 메서드를 호출하면 됨
  - ✓ 스레드가 종료할 때까지 대기는 join()
  - ✓ time 모듈의 sleep(시간)은 시간 동안 CPU를 양도하고 대기하는 메서드

Thread

- Thread
  - ✓ 일반 함수 호출: 일반 적인 함수 호출을 연속적으로 하는 경우 하나의 함수가 종료될 때 까지 다른 함수는 수행될 수 없음

```
import threading, time

def threadEx(id):
    for i in range(10):
        print('id={0} --> {1}'.format(id, i))
        time.sleep(1)

for i in range(2):
    threadEx("{0}번 스레드".format(i))
```

Thread

- Thread
  - ✓ 스레드를 생성해서 실행

```
import threading, time

def threadEx(id):
    for i in range(10):
        print('id={0} --> {1}'.format(id, i))
        time.sleep(1)

for i in range(2):
    id = ("{0}번 스레드".format(i))
    th = threading.Thread(target=threadEx, args=(id,))  # 쓰레드 함수를 target인수로, 쓰레드로 전달될 인수를 args인수에 전달
    th.start()
print("메인 종료")          # 쓰레드 실행 시작
```

Thread

- Thread
  - ✓ Thread 클래스를 상속받아서 생성

```
import threading, time

class ThreadEx(threading.Thread):
    def run(self):
        for i in range(10):
            print('id={0} --> {1}'.format(self.getName(), i))
            time.sleep(1)

for i in range(2):
    th = ThreadEx()
    th.start()
```

Thread

- MultiThread
  - ✓ 어느 한 시점에 2개 이상의 Thread가 동작 중인 경우
  - ✓ 하나의 프로그램에서 여러 개의 스레드를 사용하게 되면 프로그램의 성능은 우수해 질 가능성이 높지만 하나의 자원을 여러 개의 스레드가 공유해서 사용하는 경우에 문제가 발생할 가능성이 있습니다.
  - ✓ 멀티 Thread에서 발생할 수 있는 문제점
    - 임계 영역(critical section)
      - 공유 자원을 사용하는 코드 영역이 임계 영역
      - 이 부분에서는 공유 자원을 동시에 수정할 수 없도록 상호 배타적으로 실행될 수 있도록 작성해야 함 – Mutual Exclusion
    - Dead Lock
      - 멀티 Thread를 사용할 때 Thread의 수행이 이루어지지 않고 무한 대기하는 상태
    - 생산자와 소비자 문제
      - 공유 자원을 사용할 때 순서의 문제

Thread

- MultiThread
  - ✓ 공유 자원 수정 문제

```
import threading, time
g_count = 0

class ThreadEx(threading.Thread):
    def run(self):
        global g_count
        for i in range(10):
            print('id={0} 증가하기 전 --> {1}'.format(self.getName(), g_count))
            g_count = g_count + 1
            time.sleep(0.5)
            print('id={0} 증가한 후 --> {1}'.format(self.getName(), g_count))
            time.sleep(0.5)

for i in range(2):
    th = ThreadEx()
    th.start()
```

Thread

- MultiThread
  - ✓ Lock/RLock 인스턴스
    - Lock 클래스의 인스턴스
      - threading 모듈의 Lock()으로 생성
      - acquire(): Lock을 얻는 메서드로 이 메서드를 스레드가 호출하면 acquire()를 시도하는 다른 스레드는 Lock을 얻지 못하고 대기 상태에 들어가게 됨
      - release(): Lock을 해제하는 메서드로 이 메서드가 호출되면 대기 중인 스레드가 있다면 그 중에 하나가 깨어나서 대기 상태에서 빠져 나오게 됨
    - RLock 클래스는 Lock 클래스와 용도는 같으나 하나의 스레드가 acquire()를 한 번 이상 호출할 수 있다는 것이 다름

Thread

- MultiThread
  - ✓ Lock/RLock 인스턴스

```
import threading, time
g_count = 0
lock = threading.Lock()

class ThreadEx(threading.Thread):
    def run(self):
        global g_count
        global lock
        for i in range(10):
            lock.acquire()
            print('id={0} 증가하기 전 --> {1}'.format(self.getName(), g_count))
            g_count = g_count + 1
            time.sleep(0.5)
            print('id={0} 증가한 후 --> {1}'.format(self.getName(), g_count))
            lock.release()
            time.sleep(0.5)

for i in range(2):
    th = ThreadEx()
    th.start()
```

Thread

- MultiThread
  - ✓ 생산자와 소비자는 동시에 자원을 사용할 수 있지만 생산자가 물건을 생산해 주어야만 소비자가 물건을 소비할 수 있음
  - ✓ 생산자가 자원을 생성해 주지 않은 상황에서 소비자가 자원을 사용하려고 하면 자원이 없으므로 예외가 발생하는데 이러한 상황을 생산자와 소비자 문제라고 함
  - ✓ 소비자는 자원이 없는 경우에는 대기하고 있다가 생산자가 자원을 생성하면 자원 생성 여부를 소비자에게 신호를 주어서 자원이 생성되었다는 것을 알고 자원을 사용하도록 해주어야 함

Thread

- MultiThread
  - ✓ 생산자 와 소비자

```
import threading, time
shareData = []

class Producer(threading.Thread):
    def run(self):
        global shareData;
        for i in range(10):
            print("공유자원 생성")
            shareData.append(i)
            time.sleep(1)

class Customer(threading.Thread):
    def run(self):
        global shareData;
        for i in range(10):
            print("공유자원 사용")
            print(shareData.pop())
            time.sleep(1)
```

Thread

- MultiThread
  - ✓ 생산자 와 소비자

```
producer = Producer()
customer = Customer()

producer.start()
customer.start()
```

Thread

- MultiThread
  - ✓ 생산자 와 소비자
    - Condition 인스턴스를 이용해서 생산자와 소비자 문제를 해결
    - Condition 인스턴스의 wait()를 호출하면 현재 스레드는 대기 상태에 진입
    - Condition 인스턴스의 notify()를 호출하면 대기 중인 스레드 중에서 하나의 스레드가 다시 실행 상태가 됨

Thread

- MultiThread
  - ✓ 생산자 와 소비자
    - ● 생성자 와 소비자 문제 해결

```
import threading, time
shareData = []
cv = threading.Condition()
class Producer(threading.Thread):
    def run(self):
        global shareData;
        for i in range(10):
            cv.acquire()
            print("공유자원 생성")
            shareData.append(i)
            time.sleep(1)
            cv.notify()
            cv.release()
```

Thread

- MultiThread
  - ✓ 생산자 와 소비자
    - ● 생성자 와 소비자 문제 해결

```
class Customer(threading.Thread):
    def run(self):
        global shareData;
        for i in range(10):
            cv.acquire()
            if len(shareData) < 1:
                cv.wait(0)
            print("공유자원 사용")
            print(shareData.pop())
            time.sleep(1)
            cv.release()

producer = Producer()
customer = Customer()

producer.start()
customer.start()
```

Thread

- 세마포어(Semaphore)
  - ✓ 내부에 정수형 카운트 변수를 소유해서 동시에 실행할 수 있는 스레드의 개수를 설정할 수 있는 클래스
  - ✓ 생성할 때 동시에 수행할 스레드의 개수를 설정하고 작업을 시작하기 전에 acquire()를 호출해서 동작할 수 있으면 동작하고 카운터의 값을 1감소시키고 카운터의 값이 1보다 작으면 대기 상태로 진입
  - ✓ release()를 호출하면 카운터의 1증가

```
import threading, time
sema = threading.Semaphore(3)
class ThreadEx(threading.Thread):
    def run(self):
        sema.acquire()
        time.sleep(5)
        print(self.getName())
        sema.release()

for i in range(10):
    th = ThreadEx()
    th.start()
```

Network

- 용어
  - ✓ 네트워크: 장비 또는 사용자 들 간에 하드웨어와 소프트웨어를 공유할 수 있도록 서로 연결된 노드들의 모임
  - ✓ Protocol: 장비 사이의 데이터 송수신을 위한 규칙-TCP (연결형), UDP(비연결형)
  - ✓ IP 주소: 네트워크에서 컴퓨터를 구별하기 위한 주소(IPv4-32비트, IPv6-128비트)
  - ✓ IPv4: 32비트 주소 체계로 8개의 비트를 하나의 그룹으로 묶어서 .으로 구분하고 10진수로 표현합니다.(0.0.0.0 – 255.255.255.255)
  - ✓ IPv6: 128비트 주소 체계로 16비트를 하나의 그룹으로 묶어서 :: 으로 구분하고 16진수로 표현합니다. 연속된 0은 생략이 가능합니다.(0000::0000::0000::0000::0000::0000::0000::0000 – FFFF:: FFFF:: FFFF:: FFFF:: FFFF:: FFFF:: FFFF:: FFFF)
  - ✓ Loopback 주소: 자신의 장비 주소를 나타내는 주소로 IPv4에서는 127.0.0.1 이고 IPv6에서는 0::0::0::0::0::0::0::0::1
  - ✓ Port 번호: 하나의 ip 주소에서 프로세스를 구분하기 위한 번호로 0 ~ 65535 (0 ~ 1024번은 시스템이 사용하는 영역)으로 생성되며 하나의 응용 프로그램이 하나 이상의 포트 사용 가능

    Ex)시스템 예약: http:80, FTP(20, 21), SSH(22), Telnet(23), SMTP(25), DNS(53)

    응용프로그램: Tomcat(8080), Oracle(1521), MySql(3306), MSSQL(1443)..
  - ✓ TCP/IP: 인터넷상에서 호스트들을 서로 연결시키는데 필요한 프로토콜
  - ✓ url: 인터넷 상의 자원의 위치
  - ✓ uri: 특정 자원에 대한 형식이나 고유한 이름

Network

- Socket
  - ✓ 소켓: 네트워크 카드를 추상화 한 클래스
  - ✓ 파이썬에서는 socket 모듈에서 지원
  - ✓ 특정 서비스의 포트 번호 확인

    getservbyname(servicename, protocolname)
  - ✓ 예제

```
import socket
print(socket.getservbyname('http', 'tcp'))
print(socket.getservbyname('ftp', 'tcp'))
```

```
80
21
```

Network

- 소켓의 종류
  - ✓ 도메인의 종류에 따른 소켓
    - AF_INET: IPv4 소켓으로 주소 표현은 (host, port) 튜플로 처리
    - AF_INET6: IPv6 소켓으로 주소 표현은 (host, port, flowinfo, scopeid) 튜플로 처리
  - ✓ 유형에 따른 소켓의 종류
    - SOCK_STREAM: TCP 통신 소켓
    - SOCK_DGRAM: UDP 통신 소켓

Network

- TCP 통신
  - ✓ TCP 통신 절차: 서버와 클라이언트가 TCP 소켓을 만들고 연결한 다음 데이터를 송수신하고 소켓을 종료
    - 서버
      1. TCP 소켓 인스턴스 생성

         from socket import *

         svrsock = socket(AF_INET, SOCK_STREAM)
      2. 소켓을 호스트 컴퓨터의 포트에 연결

         svrsock.bind((HOST, PORT))
      3. 한번에 처리할 수 있는 연결 수 설정

         svrsock.listen(1)
      4. 클라이언트로부터 소켓 연결이 올 때 까지 대기

         conn, addr = svrsock.accept()
      5. 클라이언트의 connect 함수로부터 소켓이 send(bytes)와 recv(버퍼사이즈)를 이용하여 데이터를 주고받음
      6. 클라이언트와의 연결 종료

         conn.close() svrsock.close()

Network

- TCP 통신
  - ✓ TCP 통신 절차: 서버와 클라이언트가 TCP 소켓을 만들고 연결한 다음 데이터를 송수신하고 소켓을 종료
    - 클라이언트
      1. TCP 소켓 인스턴스 생성

         ```
         from socket import *
         clientsock = socket(AF_INET, SOCK_STREAM)
         ```

      2. 소켓을 서버 소켓에 연결

         ```
         clientsock.connect((HOST, PORT))
         ```

      3. 데이터 수신 및 전송

         clientsock.recv(1024)나 clientsock.send(데이터)를 이용해서 데이터 수신 및 전송

      4. 연결 종료

         ```
         clientsock.close()
         ```

Network

- TCP 통신
  - ✓ TCP 통신
    - ● 서버

```
#서버 코드
from socket import *
try:
    svrsock = socket(AF_INET, SOCK_STREAM)
    svrsock.bind(('localhost', 8000))
    svrsock.listen(1)
    print("서버 대기 중")
    conn, addr = svrsock.accept()
    print(addr)
    b = conn.recv(1024)
    conn.send('Hello Client'.encode())
    print(b.decode())
except Exception as e:
    print("예외가 발생했습니다.", e)
finally:
    conn.close()
```

Network

- TCP 통신
  - ✓ TCP 통신
    - ● 클라이언트

```
#클라이언트 코드
from socket import *
try:
    sock = socket(AF_INET, SOCK_STREAM)
    sock.connect(('localhost', 8000))
    sock.send('Hello Server'.encode())
    b = sock.recv(1024)
    print(b.decode())
except Exception as e:
    print("접속 에러:", e)
finally:
    sock.close()
```

Network

- UDP 통신
  - ✓ UDP 소켓 프로그래밍은 TCP와 약간 다른 절차로 진행
  - ✓ 소켓 인스턴스는 AF_INET과 SOCK_DGRAM으로 생성
  - ✓ 연결 요청과 접수 과정이 필요가 없고 서버는 사용하려는 포트 번호로만 소켓을 묶으면 되고 클라이언트도 자신이 사용하려는 포트 번호를 소켓에 묶으면 됨
  - ✓ 데이터 전송과 수신은 sendto() 와 recvfrom()으로 이루어 짐
  - ✓ 보내는 메서드: address는 host와 port의 튜플

    sendto(bytes[, flags[, address)
  - ✓ 읽어 내는 메서드: 리턴하는 데이터는 데이터와 클라이언트 주소 정보의 튜플

    recvfrom(bufsize[, flags])

Network

- UDP 통신
  - ✓ 서버 코드

```
from socket import *
try:
    svrsock = socket(AF_INET, SOCK_DGRAM)
    svrsock.bind(('localhost', 8001))
    while True:
        print("서버 대기 중...")
        s, addr = svrsock.recvfrom(1024)
        print("접속한 곳:{0}".format(addr))
        print("메시지:" + s.decode())
        svrsock.sendto(s, addr)
except Exception as e:
    print("예외 발생:", e)
finally:
    svrsock.close()
```

Network

- UDP 통신
  - ✓ 클라이언트 코드

```
from socket import *
try:
    sock = socket(AF_INET, SOCK_DGRAM)
    msg = input("보낼 메시지:")
    sock.sendto(msg.encode(), ('localhost', 8001))
    s, addr = sock.recvfrom(1024)
    print(s.decode())
    print(addr)
except Exception as e:
    print("접속 에러:", e)
finally:
    sock.close()
```

Network

- 소켓의 동작 모드 와 타임 아웃
  - ✓ 소켓은 블로킹 모드로 동작
  - ✓ 블로킹 모드에서는 accept()와 recv(), send() 메서드를 호출했을 때 연결하고자 하는 클라이언트가 없거나 읽을 데이터가 없는 경우 또는 보낼 데이터를 즉시 보낼 수 없을 때 그것을 처리할 때 까지 대기
  - ✓ 비블로킹 모드에서는 accept(), recv(), send() 메서드를 호출할 때 연결하고자 하는 클라이언트가 없거나 읽을 데이터나 보낼 데이터가 없으면 socket.error 예외가 발생
  - ✓ 소켓의 setblocking(True 또는 False)로 설정
  - ✓ 소켓이 일정 시간 동안 기다리도록 설정

    settimeout(초)

Network

- Broadcast
  - ✓ 동일한 네트워크에 속한 모든 인스턴스와 통신하는 모델
  - ✓ 소켓을 만들고 옵션을 설정

    소켓.setsockopt(SOL_SOCKET, SO_BROADCAST, 1)
  - ✓ 데이터 전송

    소켓.sendto(전송할 바이트, (전송할 네트워크 대역, 포트번호))

Network

- Broadcast
  - ✓ 서버

```
#서버 - 나중에 실행 - 자신의 아이피를 확인해서 아이피는 수정하세요
from socket import *
try:
    svrsock = socket(AF_INET, SOCK_DGRAM)
    #브로드 캐스팅 옵션 설정
    svrsock.setsockopt(SOL_SOCKET, SO_BROADCAST, 1)
    svrsock.sendto('Broadcasting Message'.encode(), ('192.168.0.255', 8000))
    print("전송 성공")
except Exception as e:
    print("접속 에러:", e)
finally:
    svrsock.close()
```

Network

- Broadcast
  - ✓ 클라이언트

```
from socket import *
try:
    sock = socket(AF_INET, SOCK_DGRAM)
    sock.bind(('', 8000))
    msg, addr = sock.recvfrom(1024)
    print(msg.decode())
    print(addr)
except Exception as e:
    print("접속 에러:", e)
finally:
    sock.close()
```

Network

- 채팅
  - ✓ 채팅 서버

```
+++ 채팅 서버를 시작합니다.
+++ 채텅 서버를 끝내려면 Ctrl-C를 누르세요.
[127.0.0.1] 연결됨
+++ 대화 참여자 수 [1]
[127.0.0.1] 연결됨
+++ 대화 참여자 수 [2]
뭐하니
밥은 먹고 다니니
```

Network

- 채팅
  - ✓ 채팅 서버

```
#채팅 서버
import socketserver
import threading

HOST = ''
PORT = 9009
lock = threading.Lock()
```

Network

- 채팅
  - ✓ 채팅 서버

```
class UserManager:
    def __init__(self):
        self.users = {}

    def addUser(self, username, conn, addr):
        if username in self.users:
            conn.send('이미 등록된 사용자입니다.₩n'.encode())
            return None

        # 새로운 사용자를 등록함
        lock.acquire()
        self.users[username] = (conn, addr)
        lock.release()

        self.sendMessageToAll('[%s]님이 입장했습니다.' % username)
        print('+++ 대화 참여자 수 [%d]' % len(self.users))

        return username
```

Network

- 채팅
  - ✓ 채팅 서버

```
def removeUser(self, username):
    if username not in self.users:
        return

    lock.acquire()
    del self.users[username]
    lock.release()

    self.sendMessageToAll('[%s]님이 퇴장했습니다.' % username)
    print('--- 대화 참여자 수 [%d]' % len(self.users))
```

Network

- 채팅
  - ✓ 채팅 서버

```
def messageHandler(self, username, msg):
    if msg[0] != '/':
        self.sendMessageToAll('[%s] %s' % (username, msg))
        return

    if msg.strip() == '/quit':
        self.removeUser(username)
        return -1

def sendMessageToAll(self, msg):
    for conn, addr in self.users.values():
        conn.send(msg.encode())
```

Network

- 채팅
  - ✓ 채팅 서버

```
class MyTcpHandler(socketserver.BaseRequestHandler):
    userman = UserManager()

    def handle(self):
        print('[%s] 연결됨' % self.client_address[0])

        try:
            username = self.registerUsername()
            msg = self.request.recv(1024)
            while msg:
                print(msg.decode())
                if self.userman.messageHandler(username, msg.decode()) == -1:
                    self.request.close()
                    break
                msg = self.request.recv(1024)
        except Exception as e:
            print(e)

        print('[%s] 접속종료' % self.client_address[0])
        self.userman.removeUser(username)
```

Network

- 채팅
  - ✓ 채팅 서버

```
def registerUsername(self):
    while True:
        self.request.send('로그인ID:'.encode())
        username = self.request.recv(1024)
        username = username.decode().strip()
        if self.userman.addUser(username, self.request, self.client_address):
            return username
```

Network

- 채팅
  - ✓ 채팅 서버

```
class ChatingServer(socketserver.ThreadingMixIn, socketserver.TCPServer):
    pass

def runServer():
    print('+++ 채팅 서버를 시작합니다.')
    print('+++ 채텅 서버를 끝내려면 Ctrl-C를 누르세요.')

    try:
        server = ChatingServer((HOST, PORT), MyTcpHandler)
        server.serve_forever()
    except KeyboardInterrupt:
        print('--- 채팅 서버를 종료합니다.')
        server.shutdown()
        server.server_close()

runServer()
```

Network

- 채팅
  - ✓ 채팅 클라이언트

```
import socket
from threading import Thread

HOST = 'localhost'
PORT = 9009

def rcvMsg(sock):
    while True:
        try:
            data = sock.recv(1024)
            if not data:
                break
            print(data.decode())
        except:
            pass
```

Network

- 채팅
  - ✓ 채팅 클라이언트

```
def runChat():
    with socket.socket(socket.AF_INET, socket.SOCK_STREAM) as sock:
        sock.connect((HOST, PORT))
        t = Thread(target=rcvMsg, args=(sock,))
        t.daemon = True
        t.start()

        while True:
            msg = input()
            if msg == '/quit':
                sock.send(msg.encode())
                break

            sock.send(msg.encode())

runChat()
```