

FEB_First

Thur

**Exception Handling(예외처리)

< 오류의 종류 >

< Compile Error >

: 문법적인 오류 (Compile 오류)

Compile = 물리적 오류.
Tip : Type을 확인.

특징

- 1. 물리적인 오류라서 Application 이 실행되지 않음
때문에 물리적인 오류 부분을 수정해서 실행해야 한다
- 2. 값을 확인하는 것이 아닌 Type 을 확인한다.

실행은 안됨.

* Compile

: 소스코드를 운영 체제나 VM(Virtual Machine) 등이 이해할 수 있는 코드로 변환하는 작업

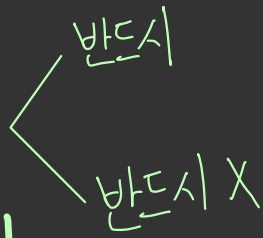
< Exception(예외) >

: 문법적인 오류가 아니기 때문에
Application 이 실행은 되지만
실행 도중 특정한 상황이 발생하면
Program이 중단되는 현상

Exception

: 실행은 된다.

논리적 오류



Ex)

URL 을 문자열로 입력해야하는 상황
" " 안에 어떤 문자열이든지 대입 시 Compile Error 가 발생하지 않음

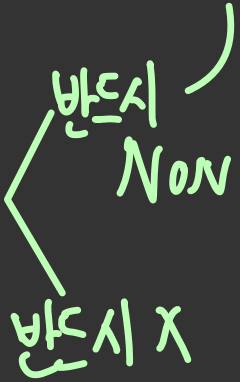
-> 유효하지 않은 URL 을 설정하면 실행하다
잘못된 URL 로 인해서 Application이 중단

Try catch

예외의 종류

- 1. 예외처리를 반드시 해야만 하는 예외(Non Runtime Exception)
- 2. 예외 처리를 하지 않아도 되는 예외 (Runtime Exception)
-> 예외 처리를 하지 않았음에도 발생하는 예외

예외처리



* Spring JDBC 의 예외 장점

예외 처리를 자바 Web Application 을 할 때 보다 적게한다

< Logic Error > (논리적 오류)

정의

: 잘못된 알고리즘으로 인한 결과 오류

이러한 오류를 줄이기 위해

- 1. Test 주도 개발을 실행
- 2. 테스트 팀을 갖추

< Assertion(단언) >

정의
: 문법적인 오류나 예외가 발생할 상황이 아닌데
강제로 예외를 발생시키는 것을 말한다.

많이 사용되는 분야

- 1. Client Applicaiton
- 2. Server Security 분야

< 오류 발생 시 해결책 >

1. Compile Error

: 코드를 역순으로 읽어 가기
(오류를 줄이기 위해서 Code 를 자그마한 Block 단위로 작성하는 것이 좋음)

2. 예외 나 논리적 Error → Debugging 을 수행

- 1). 예외 message 를 확x인해서 Code 를 역순으로 읽어가기 코드를 봉쇄하면서 실행
- 2). White Box Test - 내부 구조를 확인 . 제어문을 확인하는 습관이 필요

3. 오류가 발생하면 오류를 로깅하고 해결했던 내용을 기록하는 것이 중요하다

: Exception 발생 시 Log 파일에 기록
다른 컴퓨터에 내용을 전송하는 것을 연습하는 것도 좋다.

< 예외 처리의 목적 >

- 예외 내용을 기록
- 정상적으로 종료하거나 예외가 발생해도 계속 작업을 수행하기 위해서

Error 발생시 Logfile 만드는 것 실행

〈 Python 제공 모듈 〉

Exception Handling

```
# 예외가 발생하는 상황

def ten_div(x):
    return 10 / x

print(ten_div(2))

print(ten_div(0))
# 해당 부분에서 Error 발생
# 0으로 나누면 Infinity

print("System Over")
# 이부분이 실행되지 않음
```

→ % 인으로
ERROR

```
/Users/mac/PycharmProjects/pythonProject/venv/bin/python /Users/mac/PycharmProjects/pythonProject/0203_5th.py
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0203_5th.py", line 9, in <module>
    print(ten_div(0))
  File "/Users/mac/PycharmProjects/pythonProject/0203_5th.py", line 4, in ten_div
    return 10 / x
ZeroDivisionError: division by zero
5.0

Process finished with exit code 1
|
```

————→ Python 의 기본적인 예외 처리 방법

try :
예외가 발생할 만한 코드
except:
예외가 발생하면 수행할 코드

› 예외 처리 적용

```
# 예외가 발생하는 상황

def ten_div(x):
    return 10 / x

try:
    print(ten_div(2))

    print(ten_div(0))
# 해당 부분에서 Error 발생
# 0으로 나누면 Infinity

except:
    print("Divided Error Emerge")

print("System Over")
# 프로그램이 정상적으로 실행이 되는 예외 처리
```

0203_5th ×

```
/Users/mac/PycharmProjects/pythonProject/venv/bin/python /Users/mac/PycharmProjects/pythonProject/0203_5th.py
5.0
```

```
Divided Error Emerge
```

```
System Over
```

```
Process finished with exit code 0
```

< 예외를 나누어서 처리 >

try:

예외가 발생할 만한 코드

except 예외 클래스 이름1 as 별명 :

예외클래스 이름 1에 해당하는 예외가 발생하면 수행할 코드

except 예외 클래스 이름 2 as 별명 :

예외클래스 이름 2에 해당하는 예외가 발생하면 수행할 코드

```
# 예외가 발생하는 상황

def ten_div(x):
    return 10 / x

try:
    print(ten_div(2))

    print(ten_div(0))
# 해당 부분에서 Error 발생
# 0으로 나누면 Infinity

# 예외가 발생했을 때 message 출력
except Exception as e:
    print(e)

print("System Over")
# 프로그램이 정상적으로 실행이 되는 예외 처리
```

Exception

```
/Users/mac/PycharmProjects/pythonProject/venv/bin/python /Users/mac/PycharmProjects/pythonProject/0203_5th.py
5.0
division by zero
System Over

Process finished with exit code 0
|
```

< 예외 처리 후 실행 코드 >

Try :
except:
else :
finally:
예외 발생 여부와 상관없이 수행되는 코드

```
# 예외가 발생하는 상황

def ten_div(x):
    return 10 / x

try:
    print(ten_div(2))

    print(ten_div(0))
# 해당 부분에서 Error 발생
# 0으로 나누면 Infinity

# 예외가 발생했을 때 message 출력
except Exception as e:
    print(e)
else:
    print("예외가 발생하지 않은 경우 수행 ")

finally:
    print("System Over")
# 프로그램이 정상적으로 실행이 되는 예외 처리
```

```
# 예외가 발생하는 상황

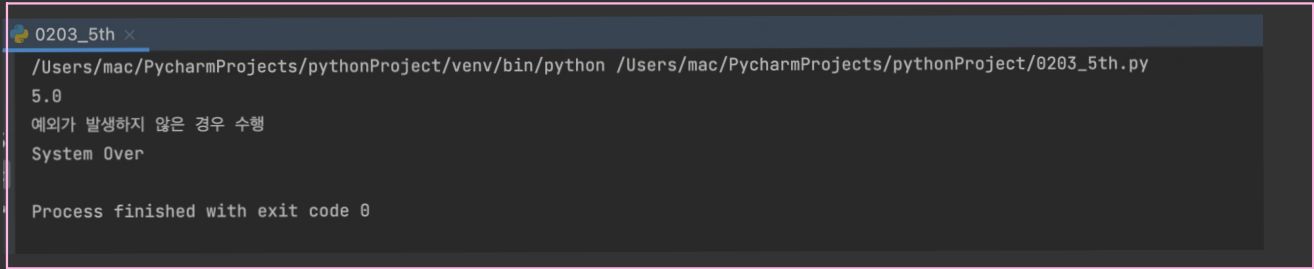
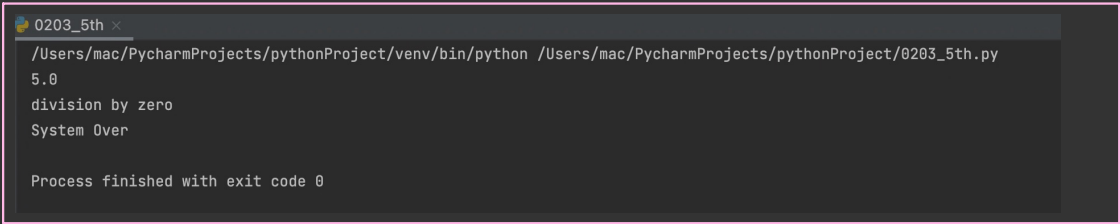
def ten_div(x):
    return 10 / x

try:
    print(ten_div(2))

    # print(ten_div(0))
# 해당 부분에서 Error 발생
# 0으로 나누면 Infinity

# 예외가 발생했을 때 message 출력
except Exception as e:
    print(e)
else:
    print("예외가 발생하지 않은 경우 수행 ")

finally:
    print("System Over")
# 프로그램이 정상적으로 실행이 되는 예외 처리
```



< 예외 강제 발생 >

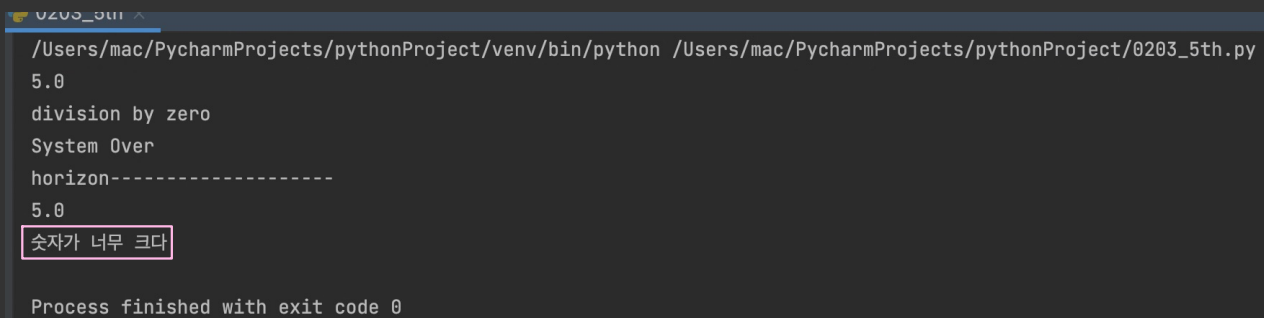
raise 예외클래스이름(message)

: 예외를 강제로 발생시켜 호출한 곳으로 예외처리를 넘기기도 한다

```
print("horizon-----")
# 예외 강제 발생
def test(x):
    if x > 10:
        raise Exception("숫자가 너무 크다")
    return 10/x

# 기본적인 예외 처리 구문을 이용해서 예외가 발생하더라도 중지되지 않도록 한다
try:
    print(test(2))
    print(test(14))
    # 예외가 발생해서 프로그램이 중단

#예외가 발생했을 때 예외 처리 message 출력
except Exception as e:
    print(e)
```



```
0203_5th ^
/Users/mac/PycharmProjects/pythonProject/venv/bin/python /Users/mac/PycharmProjects/pythonProject/0203_5th.py
5.0
division by zero
System Over
horizon-----
5.0
숫자가 너무 크다

Process finished with exit code 0
```

< 단언 >

assert 조건식 . 조건식에 맞지 않는 경우 출력할 message

→ 조건식이 False 가 되면 message 를 출력하고

AssertionError를 발생시키고 Application 은 중단

```
print("horizon-----")
score = 101
assert score <= 100, "The Score have not Over 100"
print(score)
```

```
/Users/mac/PycharmProjects/pythonProject/venv/bin/python /Users/mac/PycharmProjects/pythonProject/0203_5th.py
Traceback (most recent call last):
  File "/Users/mac/PycharmProjects/pythonProject/0203_5th.py", line 45, in <module>
    assert score <= 100, "The Score have not Over 100"
AssertionError: The Score have not Over 100
```

** Python 이 제공하는 module

< 날짜 및 시간 데이터 >

TimeStamp: 컴퓨터가 시간을 표현하는 방법

초단위, 밀리 초 단위로 측정된 시간

기준점을 1970 1 월 1 일 자정으로 본다 (Epoch Time)

* GMT : 그리니치 천문대의 평균 태양시

* UTC : 세슘 원자의 진동수에 의한 초의 길이

* LST : UTC 으로 경도 15 도 마다 1시간 차이를 두는 것

* Mac 은 운영체제를 Clean 으로 설치를 하는 경우

시간대가 안맞는 문제가 발생하는 경우가 있다

Mac 은 Clean 설치를 한 경우 환경설정에서 시간대를 한 번 변경한 후

한국을 맞추면 이 문제가 해결

MySQL을 사용하는 경우

시간대역이 맞지 않아서 오류를 발생

이때 데이터베이스 연결시 Timezone 옵션을 사용

UTC로 설정해 주면 된다

struct_time

— TimeStamp 형태로 시간을 관리하는 구조체 (Sequence Instance)

구성

tm_year

tm_mon

tm_mday

tm_hour

tm_min

tm_sec

tm_wday : 요일

tm_yday : 1월 1일부터 지나온날짜

tm_isdst : 썬더 타임 적용 여부

< time 모듈 >

: 생성 과 관련된 함수

`time.time()` : 1970년 1월 1일 자정부터 지나온 시간을 float으로 리턴

`time.gmtime()`

`time.localtime()`

현재 Thread 일시 중단

`time.sleep` (초 단위 시간)

→ `time.asctime(Struct_time)` : 문자열 형식으로 리턴

→ `time.gmtime(struct_time)` : 누적된 시간을 리턴

```
print("horizon-----")
import time

print(time.time()) #float 으로 현재시간 리턴
print(time.localtime()) # struc_time 형식으로 현재 시간 리턴

time.sleep(10)

print(time.time())
```

```
horizon-----
1643855894.0550709
time.struct_time(tm_year=2022, tm_mon=2, tm_mday=3, tm_hour=11, tm_min=38, tm_sec=14, tm_wday=3, tm_yday=34, tm_isdst=0)
1643855904.060371
```

< datetime 모듈 >

→ 하위 클래스

`datetime`: 날짜 + 시간

`date` : 날짜

`time` : 시간

`timedelta` : 시간의 차이

`tzinfo` : 시간 대역 정보

→ `date` 클래스

`datetime.date`(년, 월, 일)

`datetime.today()`

→ time 클래스

datetime.time (시, 분, 초, 밀리초 , tzinfo)

→ datetime 클래스 의 date 나 time 변환

date()

time()

combine(date, time)

→ datetime 클래스 와 문자열 변환

strptime(date_string, format) : 문자열을 datetime으로 변환

strftime(format) : datetime 을 문자열로 변환

```
print("horizon-----")
# datetime 모듈
import datetime

# 현재 시간을 가지고 생성
dt = datetime.datetime.now()

# 날짜 추출
print(dt.date())

# 시간 추출
print(dt.time())

# 년도만 추출
print(dt.date().year)

# 시간만 추출
print(dt.time().hour)
```

```
horizon-----
2022-02-03
11:55:23.614664
2022
11
```

```
# 문자열로 변환
s = dt.strftime('%Y년 %m월 %d일 %H시 %M분 %S초')
print(s)
```

```
# 문자열을 가지고 datetime 만들기
dt1 = datetime.datetime.strptime("1996-03-04 15:46", "%Y-%m-%d %H:%M")
print(dt1)
```

```
--
2022년 02월 03일 12시 03분 07초
1996-03-04 15:46:00
```

```
print("horizon-----")
# 날짜 간의 차이
import datetime

dt1 = datetime.datetime.now()
dt2 = datetime.datetime(1996, 3, 4, 16)

td = dt1 - dt2
print(td)

print(td.days, "일 지남 ")
```

```
horizon-----
9466 days, 20:06:01.890364
9466 일 지남
```

〈 수치와 관련된 module 〉

→ functions 모듈 분수 와 관련된 모듈

- 생성
Function('분수')

Function('실수')

Function('분수')

→ 속성
numerator : 분자
denominator : 분모

→ method
__floor__()

__ceil__()

__round__()

```
print("horizon-----")
# fractions 모듈의 모든 내용을
# fractions 라는 이름으로 묶어서 가져오기
import fractions

result = fractions.Fraction(5, 7) +
fractions.Fraction('6/7')
print(result)

# fractions 모듈의 모든 내용을
# fractions 라는 이름으로 묶어서 가져오기
# fractions 의 모듈의 Fraction을 현재 모듈에 포함시켜서 가져오기
# fractions 를 제외하고 사용해도 된다
from fractions import Fraction
result1 = Fraction(5, 7) + Fraction('5/7')
print(result1)
```

```
horizon-----
11/7
10/7
```

연산자 중복이 되어 있어 산술 연산이 가능하다

→ decimal 모듈

decimal 모듈의 Decimal 클래스를 사용하면
float 보다 정확한 숫자 표현이 가능

Instance 생성

Decimal(정수 또는 실수 나 실수 문자열 등)

: Decimal 인스턴스와 정수는 산술연산이 가능한데 실수와는 연산이 불가능함

```
print("horizon-----")
# 실수 표현을 정확하게 해주는 모듈

from decimal import Decimal

result = 0.0
for i in range(0, 100, 1):
    result = result + 0.1
print(result)
```

```
horizon-----
9.999999999999998
```

```
#Decimal 로 변환해서 실행
k = Decimal('0.0')
for i in range(0, 100, 1):
    k = k + Decimal("0.1")

print(k)
```

```
9.999999999999998
10.0|
```

```
print((1.0-0.8) == 0.2)
print((Decimal(1.0)-Decimal(0.8)))
print(Decimal('1.0')-Decimal('0.8'))
# compare 는 호출하는 데이터가 크면 양수(0) , 같으면 0 , 작으면 음수(-1)
print(Decimal('1.0')-Decimal('0.8').compare(Decimal('0.2')))
```

```
10.0
False
0.199999999999999555910790150
0.2
0.0
```

< random >

: python 이나 난수를 사용하기 위한 module

seed(정수) : seed 를 설정

machine Learning 이나 DeepLearning 을 하면
데이터를 분할

(model을 만들기 위한 data,
model을 테스트하기 위한 데이터 ,
검증하기 위한 data)

해서 작업을 수행하느 경우가 있다
데이터를 순서대로 분할하면 위험

→ 함수

random()

: 0 ~ 1 사이의 부동 소수점 숫자를 1 개 리턴

randint(min, max)

: min ~ max 에 속한 정수를 리턴

uniform(min, max)

: min ~ max 에 속한 부동 소수점 숫자를 리턴

randrange(dep, end, range)

: 지정된 간격으로 나열된 숫자 중 하나를 리턴

guass(m, sb)

: 가우스 분포의 난수를 리턴

shuffle(sequence Instance)

: Sequeunce 를 무작위로 섞음

choice(sequence Instance)

: Sequeunce에서 임의의 아이템을 리턴 - 복원 추출(전체 경우가 시행에 관계없이 일정)

sample(sequence Instance, k)

: 시퀀스에서 k 개의 데이터를 리턴 - 비 복원 추출(전체 경우의 수가 시행에 따른 수만큼 감소)

```
print("horizon-----")
# 랜덤 모듈 - Sampling 이나 game에 많이 사용
import random

# 0~ 1 사이 실수
print(random.random())

# 1 ~ 100 사이 숫자 무작위
print(random.randint(1, 100))

# seed 고정 - 정해진 숫자가 순서대로 return
# 여기서 42 의 의미는 아무것도 없다
random.seed(42)
print(random.randint(1, 100))
```

```
horizon-----
```

```
0.4603240373064188
```

```
40
```

```
82
```

```
horizon-----
```

```
0.29899746971745067
```

```
75
```

```
82
```

```
horizon-----
```

```
0.265229442975247
```

```
90
```

```
82
```



```
print("horizon-----")
li = ["project", "have to", "finish", "as soon as", "possible"]
# 비 복원 추출 이라서
# 동일한 데이터가 추출될 수 없다
print(random.sample(li, 5))
print("horizon-----")

# 복원 추출 - 동일한 데이터가 추출 될 수 있다.
for i in range(0, 5, 1):
    print(random.choice(li))
```

```
horizon-----
['finish', 'have to', 'possible', 'project', 'as soon as']
horizon-----
project
as soon as
have to
project
finish
```

< 문자 데이터 관련 모듈 - 정규식 >

re 모듈 제공

: 자연어 처리(불용어 제거 등에 주로 이용)

. 직접 코드를 만들어서 입력데이터 유효성을 검사하는 경우에 학습을 미리하는 것이 좋다 .

1). Matching 된

. 한글자

^

~ 로 시작하는 [] 안에서는 제외하고

\$

로끝나는 [] 안에서는 \$

[]

문자의 집합

. 나 - 를 사용해서 나열하거나 범위를 설정 가능

|

또는

()

정규식을 하나의 그룹으로 묶기

ws 공백 문자

WS 공백문자가 아닌 문다

→ 반복 관련 meta 문자

0번이상

+
1회이상

?
0 번이나 1번

{숫자}
숫자만큼 반복

{숫자 1. 숫자2}
숫자 1에서 숫자 2만큼 반복

{숫자. }
숫자 이상 반복

→ **re 모듈의 match method**
문자열에 정규식에 해당하는 pattern 이 있으면
match Instance 를 반환하고
그렇지 않다면
None을 리턴

```
print("Horizon -----")
# re 정규식 모듈
import re

match = re.match('[0-9]', '1234')
print(match)
# 정규식에 해당하는 문자가 있어서 Instance 를 리턴

# 정규식에 해당하는 문자가 없기 때문에 None 을 리턴
match = re.match('[0-9]', 'ABCD')
print(match)
```

```
Horizon -----
<re.Match object; span=(0, 1), match='1'>
None
```

→ 특수 문자

\s
공백

\w
공백 문자가 아닌 문자

\d
숫자

\D
숫자가 아닌 문자

\w
숫자 또는 문자

\W
숫자 나 문자가 아닌 문자

→ flag

re.I
대소문자를 구분하지 않기

re.M
여러줄에 걸쳐서 matching

→ re 클래스 method

compile(pattern , flags)
: 정규식 Instance 생성

search(pattern , str, flags)

match(pattern , str , flags)

split(pattern, string , maxsplit=0)

sub(pattern, repl, string, count=0)
: pattern을 찾아서 repl로 치환

```
print("Horizon-----")
import re

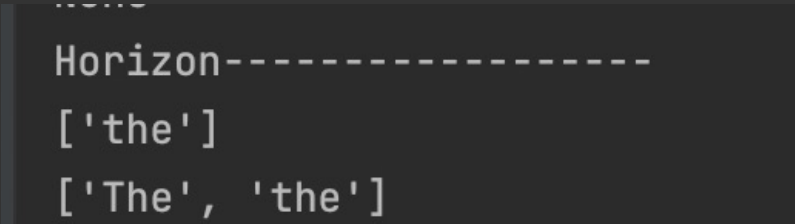
#정규식 객체 생성
# 대소문자 구분하기 때문에 'the' 만 출력
p = re.compile('the')
print(p.findall('The Hello the cat'))

# 대소문자 구분하지 않고 검색하기 때문에
# 'The' 'the' 출력
p = re.compile('the', re.I)
print(p.findall("The Hello the cat"))

# 대소문자 구분하지 않고 검색하기 때문에
# 'The' 'the' 출력
p = re.compile('the', re.I)
print(p.findall("The Hello the cat"))

# 주민등록번호 패턴 검색
p = re.compile("(\\d{6})-?(\\d{7})")
num = '123456-1234567'
if p.search(num) != None:
    print("올바른 주민등록번호 형식입니다.")
else:
    print("올바른 주민등록번호 형식이 아닙니다")

# 불용어 제거
result = re.sub("-", "", num)
print(result)
```



< 파일 시스템 관련 모듈 >

→ `os.path`

: 파일 경로를 생성 및 수정하고 파일 정보를 다룰 수 있게 해주는 모듈

`os.path.abspath(path)`

: 현재 경로를 prefix로 해서 입력받은 경로를 절대 경로로 변경해서 return

`os.path.exists(path)`

: path 의 존재여부를 return

`os.path.getatime(path)`

: 최근 접근 시간

`os.path.getmtime(path)`

: 최근 수정 시간

`os.path.getctime(path)`

: 최근 생성 시간

`os.path.getsize(path)`

: 파일 크기

`os.path.getsize(path)`

: 파일 크기

```
print("Horizon-----")

import os.path
import time
print("최종 수정 시간 : ", os.path.getmtime(
    '/Users/mac/Desktop/image'
))
print("최종 수정 시간 : ", time.gmtime(os.path.getmtime(
    '/Users/mac/Desktop/image'
)))
```

```
Horizon
최종 수정 시간 : 1642751893.4141197
최종 수정 시간 : time.struct_time(tm_year=2022, tm_mon=1, tm_mday=21, tm_hour=7, tm_min=58, tm_sec=13, tm_wday=4, tm_yday=21, tm_isdst=0)
```

→ Server에서 자주 변하지 않는 데이터를
클라이언트에 다운로드 해서 사용할 때의 알고리즘

파일의 존재 여부를 확인해서 파일이 없으면 다운로드
파일이 존재한다면 파일의 크기나 수정 날짜를 확인
→ 크기가 변경되었거나 수정날짜가 서로 다르면 다운로드

→ glob
: 디렉토리 내의 파일 및 디렉토리를 순회하고자 할 때 사용하는

glob(path)
: path 내의 파일 및 디렉토리의 list 를 리턴
path에 wildcard 문자 사용이 가능하다

iglob(path)
: path 내의 파일 및 디렉토리를 순회할 수 있는 iterator 를 리턴
→ 특정 디렉토리 내의 모든 내용을 사용하고자 할 때 사용

< 운영체제 관련 모듈 >

→ os 모듈
getcwd()
: 현재 작업 디렉토리 리턴

chdir(path)
: path 로 작업 디렉토리 변경

→ sys 모듈
prefix
: 파이썬 설치 경로

exit(정수)
: 종료 (0 이면 정상 종료 , 그 이외의 정수는 비정상 종료)

path
: 모듈의 참조 순서

getdefaultencoding()

getrefcount(instance)
: 인스턴스의 참조 횟수

```

print("Horizon-----")
# 운영체제 관련 모듈
import os
import sys
# 다른곳에서 파이선 실행시 가정 먼저 확인하는 정보

# 현재 작업 디렉토리 확인
print("현재 작업 디렉토리 확인")
print(os.getcwd())
print()

# 현재 Encoding 방식
print("현재 Encoding 방식")
print(sys.getdefaultencoding())
print()

# 현재 참조하는 모듈의 순서
print("현재 참조하는 모듈의 순서")
print(sys.path)
print()

```

```

Horizon-----
현재 작업 디렉토리 확인
/Users/mac/PycharmProjects/pythonProject

현재 Encoding 방식
utf-8

현재 참조하는 모듈의 순서
['/Users/mac/PycharmProjects/pythonProject', '/Users/mac/PycharmProjects/pythonProject', '/Library/Frameworks/Python.framework/Versions/3.9/lib/python3.9.zip',

```

화이팅

< Copy >

파이썬은 기본적으로 모든 자료형이 참조형이다
(Value 형이 없다)

대신

Scala Data : 값을 직접 참조하는 형태 , 값을 변경할 수 없다 .
Scala data를 가리키는 변수에
데이터를 수정하면 데이터가 수정되는 것이 아니고
data를 다른 곳에 저장하고 가리키는 개념

—


```

print("Horizon-----")
print("COPY")

# Scala 데이터 참조
a = 1
print(id(a))
print()

a = 2
print(id(a))
print()

# a에 저장된 데이터가 다르기 때문에 다른 id 값을 가진다

# 저장해 둔 1 을 가리키는 것이기 때문에 a가 1 일때 id 값과 동일하다
b = 1
print(id(b))
print()

# 지금 이 code 에서
a = 10
# b 는 10 이되고
b = a

# a 는 20 을 가리키기 때문에
a = 20

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

# 가리키는 위치가 다르다
print(id(a) == id(b))

# c또한 가리키는 곳이 10 이고 , b 도 10을 가리키기 때문에 id 값이 일치한다.
print(id(c) == id(b))

```

```

Horizon-----
COPY
140541262227760

140541262227792

140541262227760

140541262228368
140541262228048
140541262228048
False
True

```

→ vector data

: 참조를 복사하는데

참조를 이용해서 세부 데이터를 변경하면 동일한 참조를 가지고 있어도
데이터가 영향을 받는다.

```
print("Horizon-----")
print("Python 의 Vector ")
print()

ar = [100, 200, 300]
br = ar
print()

# 세부 데이터를 변경하였으므로 br 에 영향을 준다
print(ar)
print(br)
print()
ar[0] = 300
print(ar)
print(br)
print()

# ar 의 참조 자체가 변경된 것이므로 ar 과 br 은 아무런 관계가 없다
print(ar)
print(br)
print()
ar = [300, 200, 100]
ar[0] = 700
print(ar)
print(br)
```

```
Horizon-----
Python 의 Vector

[100, 200, 300]
[100, 200, 300]

[300, 200, 300]
[300, 200, 300]

[300, 200, 300]
[300, 200, 300]

[700, 200, 100]
[300, 200, 300]
```

→ copy 모듈의 copy method 는 얇은 복제를 지원하는 method

얇은 복제(재귀적으로 복제를 하지 않음)를 지원하는 method

deepcopy(재귀적으로 복제)

: 깊은 복제를 지원하는 method

```

print("Horizon-----")
print("COPY Module")
import copy

ar = [100, 200, 300]

# 얽은 복제를 이용 - 새로운 곳에 복사를 해서 그 곳의 id를 리턴
br = copy.copy(ar)
print(ar)
print(br)

print('middle line----')
# 새로 복제를 했기 때문에 ar 을 변형시켜도 br 에는 영향이 없음
ar[0] = 800
print(ar)
print(br)

# 문제가 되는 경우
print('middle line 2 ----')
ar = [[1, 2, 3], [4, 5, 6]]

# list안에 다른 list가 존재하는 경우 copy로 복제를 해도
# 내부 데이터를 변경하면 복제된 데이터에 영향을 준다
br = copy.copy(ar)
print(ar)
print(br)
ar[0][0] = 8920
print("result -----")
print(ar)
print(br)

#list 안에 다른 list가 존재하는 경우 deepcopy 로 복제하면
# 재귀적으로 복제를 하므로 위와 같은 현상이 없어진다
br = copy.deepcopy(ar)
print(ar)
print(br)
ar[0][0] = 989999999
print("result -----")
print(ar)
print(br)

```

```

Horizon-----
COPY Module
[100, 200, 300]
[100, 200, 300]
middle line----
[800, 200, 300]
[100, 200, 300]
middle line 2 ----
[[1, 2, 3], [4, 5, 6]]
[[1, 2, 3], [4, 5, 6]]
result -----
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]
result -----
[[989999999, 2, 3], [4, 5, 6]]
[[8920, 2, 3], [4, 5, 6]]

```

Have To Study Thing

〈 Python의 memory 정리 〉

python의 인스턴스는 **reference count** 기반의 **memory 정리 기법**을 사용

garbage collection 이 **reference count**를 확인해서
Instance의 memory를 정리

처음 Instance 가 생성될 때, **reference count** 는 1이 되고
이 Instance 를 다른 변수가 참조하면 **reference count** 는 1 이 증가

Instance를 가리키는 변수가 없어지거나
다른 인스턴스를 가리키면 **reference count** 는 1 이 감소

reference count가 0 이 되면 Instance memory 영역은 자동 소멸

```
print("Horizon-----")
print("Memory 정리")
```

```
class Temp:
    # Instance 가 memory에서 소멸될 때 호출되는 함수 --> 소멸자
    def __del__(self):
        print("Temp 클래스의 인스턴스가 파괴 된다.")
```

```
# 인스턴스가 생성 reference count 가 1
obj = Temp()
```

```
# 인스턴스를 다른 변수가 가리키도록 하는 것
# obj1 = obj
# reference count 가 1 증가한다
```

```
obj = 1
# 이전에 만들어진 인스턴스 대신에 새로 만들어진 인스턴스를 가리킨다
# 이전에 만들어진 인스턴스의 reference count 는 1 감소한다 .
```

→ weakref 모듈의 ref 함수를 이용해서 참조를 복사하면

```
print("middle-----")
#참조 횟수가 1
obj = Temp()
```

```
# 참조를 복사했으므로 참조횟수 2
obj1 = obj
```

```
# 다른 인스턴스를 참조했으므로 참조횟수가 1 줄어들어 1이 된다.
obj = Temp()
```

< Queue 모듈 >

Queue : FIFO(First In First Out)

python에서는 queue 모듈에서 Queue, Priority Queue, Stack 을 제공

queue 모듈의 클래스들은

멀티 thread 환경에서 안전하도록 설계되어 있기 때문에
동시에 데이터를 저장하고 꺼내도 정상적으로 작동하는 것을 보장

```
print("----- Horizon")
print("Queue 모듈 ")
import queue

li = ['a', 'b', 'c', 'd']

# Que를 생성해서 데이터를 추가
alphabet = queue.Queue()

for x in li:
    alphabet.put(x)

print(alphabet)

# que 에서 꺼낼때는 get() 을 사용
print(alphabet.get(0))
print(alphabet.get(0))
print(alphabet.get(0))
print(alphabet.get(0))

print('----- PriorityQueue = 우선 순위 큐 ')
li1 = ['다', '하', '사', '가']

hangle = queue.PriorityQueue()
for x in li1:
    hangle.put(x)
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))

print('-----LifoQueue = 스택 ')
li2 = ['다', '하', '사', '가']

hangle = queue.LifoQueue()
for x in li2:
    hangle.put(x)
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
print(hangle.get(0))
```

```
----- Horizon
Queue 모듈
<queue.Queue object at 0x7ff2b007f640>
a
b
c
d
----- PriorityQueue = 우선 순위 큐
가
다
사
하
-----LifoQueue = 스택
가
사
하
다
```

< Collections 모듈 >

→ Counter Class 이용

: 데이터의 개수 나 집계를 하는 것이 편리
(Dictionary 처럼 사용)

* Word Cloud를 사용하기 위해서 문장에서 나타나는 단어의 개수를 셀 때 많이 사용

```
print("----- Collection 모듈의 Counter class 를 이용한 집계 ")
from collections import Counter

portfolio = [
    ("one", 1, "일"),
    ("two", 2, "이"),
    ("three", 3, "삼"),
    ("two", 2, "이"),
    ("three", 3, "삼"),
    ("two", 2, "이"),
    ("three", 3, "삼"),
    ("two", 2, "이"),
    ("three", 3, "삼"),
    ("two", 2, "이"),
    ("three", 3, "삼"),
]

total_shares = Counter()

# 데이터 순회
for alpha, fig, han in portfolio:
    total_shares[alpha] += 1

print(total_shares)
```

```
----- Collection 모듈의 Counter class 를 이용한 집계
Counter({'two': 5, 'three': 5, 'one': 1})
```

< Thread >

독립적으로 memory 를 할당받아서 수행되는 작업의 단위

작업 도중 쉬는 시간이 생기면 다른 thread 에게 제어권을 넘겨줄 수 있다

Process 형태의 함수를 실행하는 작업은 완료되기 전에는 제어권을 넘겨줄 수 없다

→ Thread 는 하나의 Process 안에서 만들어져서 실행되고 종료

→ Thread Test 를 할 때 알아야 할 함수

Thread 쉬는 시간 생성

: time 모듈의 Sleep 함수

→ Thread 생성

: threading Thread 라는 클래스를 이용

callback 함수를 지정하고, 이용하는 방식

Thread 클래스의 초기화 함수에 target에 thread 로 동작할 함수를 지정하고

args 에 튜플 형태로 함수에 전달할 parameter 를 생성해주면 된다

thread 클래스를 상속 받는 방법

Thread 클래스를 상속받아서 run method를 재정의하면된다

→ Thread 시작

: start() method 호출

→ 다른 thread 가 종료될 때 까지 대기

: join()

```
print("Thread -----")
# 일반 함수 호출
import threading, time

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

for j in range(0, 2, 1):
    threadEx("{0}번 thread".format(j))

print("CallBack 을 이용한 함수를 지정한 thread 생성 및 시작 호출 ")
import threading, time
def threadEx(id):
    for k in range(0, 10, 1):
        print('id={0}--->{1}'.format(id, k))
        time.sleep(1)

for i in range(2):
    arg = "{0}번 thread".format(i)
    th = threading.Thread(target=threadEx, args=(arg, ))
    th.start()
```

```
Thread -----
id=0번 thread--->0
id=0번 thread--->1
id=0번 thread--->2
id=0번 thread--->3
id=0번 thread--->4
id=0번 thread--->5
id=0번 thread--->6
id=0번 thread--->7
id=0번 thread--->8
id=0번 thread--->9
id=1번 thread--->0
id=1번 thread--->1
id=1번 thread--->2
id=1번 thread--->3
id=1번 thread--->4
id=1번 thread--->5
id=1번 thread--->6
id=1번 thread--->7
id=1번 thread--->8
id=1번 thread--->9

CallBack 을 이용한 함수를 지정한 thread 생성 및 시작 호출
id=0번 thread--->0
id=1번 thread--->0
id=0번 thread--->1id=1번 thread--->1

id=0번 thread--->2
id=1번 thread--->2
id=0번 thread--->3
id=1번 thread--->3
id=0번 thread--->4
id=1번 thread--->4
id=0번 thread--->5
id=1번 thread--->5
id=1번 thread--->6
id=0번 thread--->6
id=1번 thread--->7
id=0번 thread--->7
id=1번 thread--->8
id=0번 thread--->8
id=1번 thread--->9
id=0번 thread--->9
```

```

print("상속을 이용한 thread 생성 및 시작 ")
import threading, time

class ThreadEx(threading.Thread):
    def run(self):
        for i in range(0, 5, 1):
            print('id={0}--->{1}'.format(self.getName(), i))
            time.sleep(1)

for i in range(0, 2, 1):
    th = ThreadEx()
    th.start()

```

```

----- Collection 모듈의 Counter class 를 이용한 집계
상속을 이용한 thread 생성 및 시작
id=Thread-1--->0
id=Thread-2--->0
id=Thread-1--->1id=Thread-2--->1

id=Thread-1--->2
id=Thread-2--->2
id=Thread-1--->3id=Thread-2--->3

id=Thread-1--->4
id=Thread-2--->4

```

< MultiThread >

- 2개 이상의 thread 가 동작 중인 경우

critical section (임계 영역)

: 공유 자원을 사용하는 코드 영역

mutual exclusion (상호 배제)

: 하나의 thread 가 사용중인 다른 thread 를 수정하면 안된다.
(동기화 문제가 발생할 수 있기 때문)

생산자와 소비자 문제

: 생산자가 공유 자원을 생성하기 전에 소비자가 자원을 사용하려 해서 발생하는 문제
—> 소비자 thread 가 종료

—> **Dead Lock**

: 공유 자원을 소유한 채로 다른 thread 가 가진 공유자원을
서로 요청해서 thread 가 더이상 수행되지 못하고 멈추는 현상을 말함

—> **semaphore**

: 여러개의 공유 자원을 관리하기 위한 알고리즘

FRI

Back End 나 Front End 개발에서는 Thread 개발이 중요

Operator(운영), Data Minining을 하고자 할 때는 Thread 의 개념이 중요