

PYTHON

내부데이터 : local

외부데이터 : local 밖

media : web 이나 다른 곳

DATA Analysis

: 비지니스 인사이트를 도출하기 위한 목적

컴파일(compile) : 실행되는 코드를 만들지 않음 (중간 코드 // java : bytecode)

인터프리터(interpreter) : 해석 하면서 실행 // Debugging 이 어렵다

python

장점 : 생태계가 좋음

// 자바보다 빨리 만들어짐

- 플랫폼에 독립적인 언어

Edge Computing 을 생각해보자

— Client 에서 DATA 처리를 하는 것

: 보안이 문제가 됨

: 데이터를 서버에 보낼 수 없는 상황이 있고.

자바스크립트에서 난독화 중요

Python

— 개발환경 구성

1. python 번역기 사용

- Local 에 설치해서 사용
- Cloud 에서 사용 - Goolge 의 Colab

2. Python 배포판 설치

- python
- : 기본 배포판

- anaconda
- : 선형 대수 나 기술 통계 분석 , 머신 러닝 용 라이브러리가 포함된 버전

3. anaconda 설치

4. IDE

————— python 을 설치하면 제공되는 IDE

- python console
- python IDLE

————— anaconda 를 설치하면 제공되는 IDE

ipython

spyder

jupyter notebook : 브라우저에서 동작, 별도로 설치가능

————— 별도로 설치해서 사용할 수 있는 IDE

Eclipse 의 PyDev plugin

Pycharm

: 응용 프로그램 개발에 많이 사용

VS Code

클라우드 플랫폼 : Google 의 colab , 구름 , Programmers

부가적으로 설치

- 자연어 처리를 하고자 하는 경우 Java 를 설치하는 것이 좋다 .
한국어 자연어 처리에 많이 사용하는 Konlpy 가 Java 로 만들어져 있기 때문
- windows 에서 할 때는 visual Studio 재배포 package를 설치하는 것이 좋다 .
python 도 open source 진영에서 많이 사용
python 라이브러리 중에 c 언어로 만든 것 중에서 windows에서
바로 배포 되지 않는 경우가 있기 때문에

Python의 구성요소 —> 프로그램의 구성요소라고 봐도 됨

1. Literal : 사용자가 입력하는 데이터
2. Variable : 데이터를 저장한 공간에 붙인 이름
3. Function : 코드를 이름만으로 사용 할 수 있도록 묶어 놓은 것
호출하게 되면 별도의 메모리 영역을 할당받아서 수행
수행이 종료되면 메모리는 반환
4. Class 와 Instance : 자주사용하는 변수와 함수를 묶어 놓은것
5. Module : 하나의 파일을 지칭
6. Package : 관련있는 module 의 모임
압축된 형태나 디렉토리 형태로 존재하는 배포의 단위
7. comment : Python 번역기가 번역하지 않는 문장

Python Coding

: block을 들여쓰기를 이용해서 정의
기본 공백 4칸 인데 일정한 공백만 사용하면 칸 수는 상관없다.

- Block 을 만들어서 하위 블록 생성시
상위 블록의 마지막에 : 을 추가해야한다.
- 줄 단위로 번역해서 실행하기 때문에 명령어를 구분하는 기호는 없음
- 한 줄에 2개 이상의 명령이 작성되었을 때는 ; 로 구분
- 마지막 문장에 변수 이름이나 표현식 또는 리턴이 있는 함수를 호출하면 결과는 출력

```
import random

random.randint(1, 20)
num = random.randint(1, 20)
# 파이썬은 하위 블록을 만들 때 : 을 붙이고 일정한 크기만큼 들여쓰기를 해야한다.
if num >= 10:
    print(num)
    print("10 보다 크다 ")
else:
    print(num)
    print("10보다 작다.")
```

```
#변수를 맨 마지막에 입력해도 출력이됨
# 여기서 출력되는 값은 print(x)
x=10
print(x)
# 있어도 Error가 발생하지 않음
x
```

10

```
x=10
print(x)
# 있어도 Error가 발생하지 않음
x
print("another command")
```

10

another command

Comment

- 파이썬은 주석을 만들 때 앞에 # 을 추가하면 된다
- 특정 영역의 코드를 실행하지 않도록 만들 때
대신에 """ 이나 ''' ''' 으로 묶는 경우도 있다
(이 경우 주석이 아니라 문자열 리터럴을 만드는 것이다)

#!

: #! 로 시작하는 기호는
주석이 아니고 유닉스의 Shebang
이 스크립트는 프로그램으로 실행하라는 의미이다.

#-*- coding

: Encoding 인코딩 방식 -*- 는
주석이 아니라 인코딩 설정문이다.

```
'''  
이 문장은 주석이 아니고 여러 줄 문자열을 만들어서  
실행되지 않도록 한 것이다  
print("하나")  
'''
```

dir 

: 데이터를 대입하면 데이터가 사용할 수 있는 속성이나 함수 목록을 리턴

```
x = 'a'  
print(dir(x))
```

```
['__add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '__eq__', '__format__',  
 '__ge__', '__getattr__', '__getitem__', '__getnewargs__', '__gt__', ...]
```

help

: 함수 이름 대입 시 함수에 대한 설명을 확인 할 수 있음

```
help(max)
print(max([1, 3, 5]))
```

max(...)

max(iterable, *[, default=obj, key=func]) -> value

max(arg1, arg2, *args, *[, key=func]) -> value

With a single iterable argument, return its biggest item. The default keyword-only argument specifies an object to return if the provided iterable is empty.

With two or more arguments, return the largest argument.

5

Keyword(예약어)

: 프로그래밍 언어가 기능을 정한 명령어

— 이 명령어들은 다른 용도로 사용할 수 없다

(Caution: Class 이름은 예약어가 아니다)

#예약어 확인 방법

```
import keyword
print(keyword.kwlist)
```

```
['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async',
'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except',
'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal',
'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']
```

Python 이 상대적으로 간결하다고 하는 이유

—> 보기처럼 예약어가 적다

python 에서 Module 을 찾는 순서를 확인

— Package 를 설치 했는데 사용할 수 없다고 한다면
확인을 해야한다 .

```
## 모듈을 찾는 순서 확인
```

```
import sys  
print(sys.path)
```

```
['/Users/mac/PycharmProjects/pythonProject', '/Users/mac/PycharmProjects/  
pythonProject', '/Library/Frameworks/Python.framework/Versions/3.9/lib/  
python3.9.zip', '/Library/Frameworks/Python.framework/Versions/3.9/lib/  
python3.9', '/Library/Frameworks/Python.framework/Versions/3.9/lib/  
python3.9/lib-dynload', '/Users/mac/PycharmProjects/pythonProject/venv/lib/  
python3.9/site-packages']
```

```
nation = "KK"  
#python의 str 을 사용하기 때문에 소문자로 변환  
print(str.lower(nation))  
  
# 내 모듈에서 str을 10으로 정의  
str= 10
```

kk

```
nation = "KK"  
#python의 str 을 사용하기 때문에 소문자로 변환  
print(str.lower(nation))  
  
# 내 모듈에서 str을 10으로 정의  
str= 10  
# str이 파이썬의 것이 아니고 내 모듈에서 str 이므로 에러  
print(str.lower(nation))
```

```
[['_add__', '__class__', '__contains__', '__delattr__', '__dir__', '__doc__', '_  
Traceback (most recent call last):  
  File "/Users/mac/PycharmProjects/pythonProject/main.py", line 58, in <module>  
    print(str.lower(nation))  
AttributeError: 'int' object has no attribute 'lower'  
Help on built-in function max in module builtins:
```

› Error 발생

파이썬의 기본 데이터 자료형

— 리터럴은 값을 변경할 수 없다.

```
#10 을 저장하고 그 위치를 x 가 가리킨 것
x = 10
# x 값을 출력
print(x)
# 저장된 데이터의 위치를 출력
print(id(x))

x = 20
print(x)
print(id(x))

y = 10
print(y)
print(id(y))
```

```
10
140562334411344
20
140562334411664
10
140562334411344
```

10 의 위치를 가리키기 때문에 값이 같다

종류

bool : true , false를 저장하는 자료형

수치형 : int , float , complex, - scala 자료형 , 변경 불가능 , 직접 접근

문자열 : str - vector 자료형 , 변경 불가능 , 시퀀스 를 이용해서 접근

tuple - vector 자료형 , 변경 불가능 , 시퀀스를 이용해서 접근

list - vector 자료형 , 변경 가능 , 시퀀스를 이용해서 접근

dict - vector 자료형 , 변경가능 , mapping 을 이용해서 접근

set - vector 자료형 , 변경 가능 , Set(중복 불가) 을 이용해서 접근

숫자 자료형 표현

정수 : 일반적인 정수는 10 진수 , 8진수(0o), 16진수(0x) , 2진수(0b)를 붙여서 표현

실수 : 1.7 형태의 고정 소수점 방식과 1.4e 지수를 이용하는 부동 소수점 방식으로 표현 가능
(지수는 정수 가능 , 소수는 정수 불가)

복소수 : 허수 부분에 j 나 J 를 붙여서 표현 $-4 + 5j$

bool

— true 와 false 로 표현

- 0 이 아닌 숫자는 true

데이터가 존재하는 vector 도 true로 간주

데이터의 집합

문자열 (str)

: 작은 따옴표, 큰따옴표안에 작성
여러 줄의 문자열을 하나의 데이터로 만드려면 따옴표 3번씩

bytes

: byte 의 집합
b '문자열' 또는 b 'w 코드 나열'
다른 곳에서 데이터를 가져올 때 사용

list

: 서로 독립적으로 비교 가능한 데이터를 하나로 묶을 때 사용
[데이터 나열]
로 표현

tuple

: 행
모여서 하나의 데이터를 표현하기 위한 자료형
(데이터 나열)
로 표현

set

: 중복 없는 데이터의 모임
{데이터의 나열}
로 표현

————— 여기까지 자료형들은 for 를 이용해서 순서대로 데이터를 접근할 수 있다고해서
iterable 이라고 한다

dict

: 키와 값을 쌍으로 저장하는 자료형
{키 : 값, 키 : 값 ... }
의 형태로 표현
({ } 안에 아무런 데이터도 없으면 dict 이다)

제어 문자

- `\w` 다음에 영문자 1자를 추가해서 특별한 의미를 부여한 문자

- `\wn` : 줄바꿈

- `\wt`: 탭 ,

- `\ww:w` ,

- `\w"`

- `\w'`

- 등이 있다

- (`\wo` 는 Null)

None

: 가리키는 데이터가 없다는 의미

NaN 이라고도 하고

null 또는 nil 이라고도 한다

(데이터 분석을 할때는 결측치(Missing Value)라고 한다)

Identifier

- 사용자 정의 명칭

- Programmer 가 기능을 정하는 명령어

- 데이터(변수), 함수, 클래스, 객체, module, package에 이름을 붙이는 것이 가능

- 영문자와 숫자, 한글 사용 가능

- 영문자나 `_` 로 시작 (숫자부터 시작, 특수문자 `x`)

- 파이썬의 예약어는 식별자로 사용할 수 없음

- (Syntax Error: invalid syntax error)

파이썬에서 데이터에 이름을 붙이는 방법

식별자 = 데이터

- 파이썬은 자료형을 기재하지 않음
 - 이름을 삭제하고자 할 때는 del 식별자를 이용
 - 여러개를 한꺼번에 생성하는 것도 가능하다
- 이름 1, 이름 2. = 데이터 1, 데이터 2

```
#singsiuk이라는 문자열에 name이라는 이름 붙이기
name = "singsiuk"
print(name)
_____
singsiuk
```

```
print(abs(-3))
: error /
abs는 절대값을 구해주는 함수의 이름인데
정수를 저장한 곳을 가리키게 되서
```

: 기능이 변경되서어러

```
abs = 7
print(abs(-3))
_____
error
```

```
#키워드인 and의 기능을 변경하려고 해서 Error 발생
and = 1
print(and)
_____
error
Invalid Syntax Error
```

Operator

- 계산을 수행해주는 부호나 명령어

(분류를 할 때 는

연산의 방식(산술 연산, 논리연산) 과

피연산자의 개수(Operand) 의 개수(Unary Binary)

등으로 분류

산술 : 숫자

논리 : true , false.

1. 할당 연산자

: =

오른쪽에 데이터가 가리키고 있는 데이터를

왼쪽의 이름이 가리키도록 해주는 연산자

2. 산술 연산자

: +

동일한 자료형의 데이터끼리만 연산이 가능

숫자의 데이터의 경우 더하고

vector 자료형은 곱합

+ 가 붙었을 때는

양쪽 데이터의 자료형을

반드시 확인해야 한다

Operator (연산자)

```
print(10+30)
# 40
print([100, 200, 300]+[1000, 1000, 1000])
#[100, 200, 300, 1000, 1000, 1000]

print([100, 200, 300]+10)
#error
#TypeError: can only concatenate list (not "int") to list

import numpy as np
li = [100, 300, 200]
ar = np.array([100, 300, 200])
print(li)
print(ar)
print(ar + 100)

#[100, 300, 200]
# [100 300 200]
# [200 400 300]
```

- (벨샵)

: 숫자 데이터에서만 사용

* (곱셈)

: 숫자 데이터끼리는 곱셈

vector 와 정수가 곱셈을 하면
vector를 정수만큼 반복

```
# Vector 와 정수가 곱셈을 하면 반복
print([1, 2, 3]*3)
#[1, 2, 3, / 1, 2, 3, / 1, 2, 3]
```

```
print( "hi!  \n"*5 )
hi!
hi!
hi!
hi!
hi!
```

—— 거듭 제곱 연산자

: 거듭 제곱 연산자

—— 실제 나눗셈 한 결과

/

: 실수로 리턴한다.

—— 나눗셈 몫 만 구하기

//

: 몫만 정수로 리턴

—— 나머지 구해주는 연산자

%

: 나머지 구해주는 연산자

등가 연산자와 항등 연산자

결과가 bool

`> , >= , < , <= , == , !=`

: 숫자 데이터와 bool 데이터 사이에도 사용할 수 있다

True 는 1 로

false 는 0으로

간주해서 수행

산술 비트 연산자

`<< , >>`

: 정수 데이터를 왼쪽 또는 오른쪽으로 밀어내는 연산자

왼 → 2배

오 → 1/2배

(자바는 32 회 이상 x
python은 32 회 이상도 가능)

```
# 왼쪽으로 2 번 --> 10 * 2^2
print(10 << 2)
40
```

```
# 오른쪽으로 2번 --> 10 * 2^-2
print(10 >> 2)
2
```


&

: 둘다 1 일 때만 1.

|

: 둘다 0 일 때만 0

^

: 두개가 같으면 0 다르면 1.

~

: 1-> 0 , 0-> 1

— 데이터 분석을 할 때 사용하는 자료구조에서
vector 자료형 끼리 연산을 하면
각 요소끼리 연산을 해서 vector로 리턴한다

정수 사이의 산술비트 연산

```
x = 21 # 이진수 10101
y = 19 # 이진수 10011
print(x & y)
# 10 --> 17 binary 10001

print(x | y)
# 10 --> 23 binary 10111

print(x ^ y)
# 10 --> 6 binary 00110
```

파이선에서는 &, | , ^ 를 정수 데이터 사이의 연산에만 사용
----- python에서는 numpy Library가 엄청 중요하다

```
import numpy as np

x = np.array([True, True, False])
y = np.array([True, True, False])

print(x & y)
```

데이터 분석 라이브러리에서는 vector 자료형에 & 이나 | , ^ 를 사용할 수 있음
이를 이용해서 조건에 맞는 데이터를 추출하는데 사용

—> [True True False]

```
x = np.array([True, True, False])
z = np.array([100, 151, 200])
print(x > 5)
# true 는 1 false 는 0 이기 때문에
—> [False False False]

print(z[z > 150])
# z 요소 중에 150 초과인 수만 출력
—> [151 200]
```