1. Programming in Python

1.1 What is data engineering

데이터 엔지니어의 주된 책임은 데이터 플랫폼을 위해 구조를 설계하고 데이터 분석가, 과학자, 다른 사람이 데이터를 효과적으로 쿼리할 수 있도록 하는 것이다. 특히 데이터 엔지니어는 데이터 에코시스템의 조각들을 연결하는 데이터 파이프라인을 구축하고 연결하는 것이다.

데이터 인프라를 구축하기위해 데이터를 모으고 처리하는것을 자동화하는 과정이 필요하다. 컴퓨터가 위의 작업을 하기위해 우리는 적절한 지시를 내려야 한다. 우리가 컴퓨터에게 어떠한 지시를 내리는 작업을 **프로그래밍**이라 한다.

컴퓨터를 프로그래밍 하기위해선 특수한 언어로 작성을 해야하는데, 이 언어를 **프로그래밍 언어** 라고 한다.

1.2 Kinds of Error

KeyError

주로 딕셔너리 자료형에서 발생하는 에러이다. 없는 Key값에 접근하려고 할 때 발생하게 된다. 이때 Dict.get() 메소드를 사용할 수 있다.

ValueError

부적절한 값을 가진 인자를 받았을 때 발생하는 에러이다. 예를들어 int함수 안에는 int("10")과 같이 숫자로 변경할 수 있는 문자열이 와야 하는데, 숫자로 변경할 수 없는 문자열이 오는 것처럼 부적절한 인자가 올 때 발생한다. 또한 리스트 자료형에서 없는 인덱스 값에 접근하려고 할 때 발생한다.

Syntax error

모든 프로그래밍언어는 고유한 문법을 가지고 있다. 이 문법을 위배해서 프로그래밍 하게되면 **문법 에러(syntax error)** 를 출력한다.

Name error

지정되지 않은 변수명을 불러오거나 변수명에 할당할 경우 **이름 에러(name error)**를 출력한다.

ZeroDivisionError

숫자를 0으로 나누려는 경우 발생한다.

FileNotFoundError

존재하지 않는 파일이나 디렉토리에 접근하려고 할때 발생한다.

TypeError

서로 다른 타입으로 연산하려고 할 때 발생한다.

AttributeError

잘못된 메서드나 속성을 호출하거나 대입했을 때 발생한다.

ConnectionError

서버를 켜지 않을 때 발생하며, 이 때 manage.py가 있는 프로젝트 디렉토리로 이동하여 python manage.py runserver를 실행한다.

Runtime error

프로그래밍 언어의 문법은 만족하지만 다른 이유로 컴퓨터가 결과를 출력하지 못할 경우 **런타임 에러(Runtime error)**를 출력한다.

1.3 Dealing Errors

EAFP

"It's easier to ask forgivness than permission"의 축약어로, 미리 허락을 구하는 것보다 나중에 용서를 구하는게 쉽다는 말이다. 에러가 발생할 모든 상황을 미리 설정하는 것이 아니라, 일단 예외가 발생하지 않을 것이라는 가정을 하고 코딩을 진행한다. 이후 에러가 발생하면 그때 예외처리를 실행하게 된다.

즉, 일단 수행(try)시키고, 에러가 발생하면 그때 처리(except)하겠다는 것이다. 파이썬에서 권장하는 방법이다.

LBYL

"Look before you leap"의 축약어로, 돌다리도 두드려 보고 건너라는 말이다. 코드를 실행하기 전에 발생할 수 있는 에러의 조건을 다 가정하고 처리할지 조건문으로 코딩하는 방법이다.

예외처리

에러가 발생하면 그 위치에서 프로그램은 종료되기 때문에 에러가 발생한 지점 이후의 코드들은 실행이 되지 않는다. 유저가 사용하는 프로그램에서 에러가 발생했을 때 프로그램이 종료되어 버린다면 큰 문제로 이어질 수 있다. 기본적으로 예외처리를 하는 문법은 다음과 같다.

- try : 에러 발생 가능성이 있는 코드 실행
- excpet : 에러 발생 시 진행되는 코드
- else : 에러가 발생하지 않았을 경우 실행 (생략이 가능하며, except가 있어야 사용할 수 있다.)
- finally : 에러가 발생해도, 발생하지 않아도 무조건 실행가능함.

1.4 Computer program

컴퓨터에게 내리는 모든 지시들을 **코드**라고 하고, 각각의 지시를 **line of code**라고 한다. 우리가 특수한 목적을 가지고 작성한 코드들을 **컴퓨터 프로그램**이라고 한다. 우리가 작성한 코드는 컴퓨터의 **input**이라고 하고, 출력된 결과를 **output**이라고 한다.

```
print(2+3)
```

1.5 Code comment

'#' 문자 뒤에 작성되는 문자열은 **주석(code comment)**라고 불린다. 주석은 주로 코드의 정보를 추가하는데 사용된다.

1.6 Arithmetics

Python에서 제공하는 기본적인 사칙연산은 +, -, *, /, ** 등이 있다.

```
print(4 * 3)
print(4 + 3)
print(4**3)
```

2. Varaibles and Data Type

2.1 Save value in variable

어떠한 연산을 통해 얻은 값을 다시 활용하기 위해 **변수(Variable)**라는 장소에 저장하게 된다. 변수에는 연산식이 저장되는 것이 아닌 연산이 된 값이 저장되게 된다. 변수는 컴퓨터 메모리의 특수한 위치에 저장되게 되며 각각의 위치는 고유의 식별자를 가지게 된다. 식별자의 이름을 **변수명**이라고 한다.

```
value = 20
```

기존 변수의 값에 추가적인 연산을 진행한 후 변수에 다시 할당하게 되면 값이 바뀌어 저장되게 된다. 변수를 재할당하는 코드는 연산자를 **할당 연산자(=)** 앞에 붙여 간략하게 사용 가능하다.

```
value = value + 30
```

2.2 Data types

컴퓨터 프로그래밍에서 서로 다른 종류의 값을 **데이터 타입**으로 분류한다. 데이터 타입은 해당 값이 어떤 방식으로 처리되어야 하는 방법을 제공하며 컴퓨터에 저장되는 방식 또한 제공한다.

숫자 데이터의 경우 하나의 데이터 타입으로 존재하는 것이 아닌 int(정수형) 자료와 float(실수형) 자료로 분리되어 사용된다. 정수에서 실수로 데이터를 변환할 경우 float() 함수를, 반대로 변경할 경우 int() 혹은 round()를 사용한다.

```
type(3)
float(3)
```

2.3 String data

파이썬의 문자열 데이터는 "" 혹은 '' 안에 표시되어 알파벳을 포함한 문자 데이터를 저장할 수 있다. 문자열 데이터를 **string**이라고 하고 데이터의 타입을 **str**로 표기한다.

2.4 Back slash

역슬래시(Backslash)은 문자 부호의 일종으로 정규 표현식에서 그 뒤에 따라 오는 문자가 특수하게 처리되어야 한다는 것을 나타낸다. 종종 탈출 문자로도 불린다.

```
motto = 'Facebook\'s new motto is "move fast with stable infra."'
```

2.5 Concatenation

복수의 문자열 데이터를 연결하는 과정을 **concatenation**이라고 한다.

```
string = "a" + "b" + "c"
print(string)
```

3. List and For loops

3.1 List

데이터 포인트(Data Point)는 정보를 제공하는 값이다. 각각의 데이터 포인터가 모여서 **데이터 셋(Dataset)**을 구성한다.

리스트(List) 자료형은 데이터셋을 저장하는 가장 기본적인 데이터 타입이다. 리스트는 데이터 타입에 관계없이 값을 저장할 수 있다. 또한 리스트 안에 있는 각 값은 상응하는 위치 인덱스를 가지게 된다. 인덱스는 0부터 시작해서 리스트의 길이보다 1만큼 작은 수에서 종료되며, 인덱스를 통해 리스트 내부의 값을 불러올 수 있다.

```
# Create row
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
row_2 = ['Instagram', 0.0, 'USD', 2161558, 4.5]

# Create dataset
data = [row_1, row_2]

# Indexing list
row_1[0]
row_1[0:3]
```

3.2 Reading files in List

리스트를 사용하면 CSV파일의 데이터를 프로그램 내부에 가져올 수 있다.

```
opened_file = open('AppleStore.csv')
from csv import reader
read_file = reader(opened_file)
opened_file.close()
apps_data = list(read_file)
```

3.3 For loops

반복문(For loop)는 반복적으로 수행해야하는 작업에 대해서 효율적으로 작업할 수 있게 도와준다.

```
row_1 = ['Facebook', 0.0, 'USD', 2974676, 3.5]
for data_point in row_1:
    print(data_point)
```

4. Conditional Statments

4.1 if/elif/else

조건문(Conditional Statement)는 특정 조건을 만족하는 데이터에 대해 코드를 작성할 때 사용된다. if ~ else 구문을 기본적으로 사용하며 복수의 조건에 대해서 if 와 else 사이에 elif를 추가할 수 있다.

```
if False:
    print(1)
elif 30 > 5:
    print('The condition above was false.')
```

4.2 Boolean type

불린 타입(boolean)데이터는 True/False로 표현되는 자료형이다. True/False로 표현되기도 하지만 >, <, ==, != 등 관계 연산자에 의해 표현되는 표현식 또한 boolean type을 가지고 있다.

4.3 Logical operator

논리 연산자(Logical Operator)는 서로 다른 boolean type 자료를 연결할 수 있다.

```
if (20 > 3 and 2 != 1) or 'Games' == 'Games':
    print('At least one condition is true.')
```

5. Functions

5.1 Creating Functions

Python에서 제공되는 기본적인 함수를 **내장 함수(built-in function)**이라 한다. 대표적인 내장함수는 sum(), max(), min() 등이 있다. **함수(Function)**은 반복적으로 수행해야 하는 작업단위를 구성하는 코드의 집합이다. 함수는 입력값을 받고, 처리하여 결과값을 출력하는 부분으로 구성되어 있다.

함수는 def statement, code block, return 파트로 구성되어 있다. def statement에서 **매개변수(parameter)**라고 불리는 입력값을 입력받는다. 매개변수에 입력되는 값을 **argumnets** 라고 한다. parameter = argument로 입력되는 방식을 **keyword argument**라고 하며, parameter 없이 argument1, argument2로 입력하는 방식을 **positional argument**라고 한다.

```
def square(number):
    return number**2

def add_to_square(x):
    return square(x) + 1000
```

5.2 Multiple return statement

```
def sum_or_difference(a, b, do_sum):
    if do_sum:
        return a + b
    return a - b

# Function return multiple results by default arguments

def open_dataset(file_name='AppleStore.csv', header=True):
    opened_file = open(file_name)
    from csv import reader
    read_file = reader(opened_file)
    data = list(read_file)

    if header:
        return data[1:]
    else:
        return data
```

5.3 Return multiple variables

```
def sum_and_difference(a, b):
    a_sum = a + b
    difference = a - b
    return a_sum, difference
sum_1, diff_1 = sum_and_difference(15, 10)

def open_dataset(file_name='AppleStore.csv', header=True):
    opened_file = open(file_name)
    from csv import reader
    read_file = reader(opened_file)
    data = list(read_file)

    if header:
        return data[1:], data[0]
    else:
        return data

apps_data, header = open_dataset()
```

6. Dictionaries

6.1 Dictionary

딕셔너리(Dictionary)는 값을 key : value 쌍으로 매핑시켜 데이터를 저장하는 자료형이다. 딕셔너리의 값은 정수, 실수, 문자열, 불린, 리스트, 딕셔너리 등 모든 자료형이 가능하다.

6.2 Creating Dictionary

```
# First way
dictionary = {'key_1': 1, 'key_2': 2}
# Second way
dictionary = {}
dictionary['key_1'] = 1
dictionary['key_2'] = 2
```

6.3 Update Dictionary

딕셔너리의 값은 mutable 하기 때문에 키의 값을 변경할 수 있다.

```
dictionary = {'key_1': 100, 'key_2': 200}
dictionary['key_1'] += 600
```

6.4 Frequency table

[value] in [dict] 문법은 딕셔너리 내부에 해당 값이 있는지 불린 자료형으로 출력한다. 반복문을 사용하면 각 열의 데이터에 대해서 빈도 테이블을 만들 수 있다.

```
opened_file = open('AppleStore.csv')
from csv import reader
read_file = reader(opened_file)
apps_data = list(read_file)
opened_file.close()

content_ratings = {}

for row in apps_data[1:] :
    c_rating = row[10]

    if c_rating in content_ratings :
        content_ratings[c_rating] += 1
    else :
        content_ratings[c_rating] = 1

print(content_ratings)
```

7. Object Oriented Programming

7.1 OOP

절차 지향 프로그래밍(Procedual Programming)은 연속적인 계산 과정을 포함한 단순 코드들의 집합이다. **객체 지향 프로그래밍(Object Oriented Programming)**은 프로그램을 단순히 데이터와 처리방법으로 구분하는 것이 아니라, 프로그램을 수많은 **객체(Object)**라는 기본 단위로 나누고 이들의 상호작용을 서술하는 방식이다.

객체(Object)는 데이터를 저장하는 엔터티이다. **클래스(Class)**는 객체의 타입을 묘사한다. **속성(attribute)**는 객체의 인스턴스에 저장되어 있는 데이터이다. **메소드(metthod)**는 객체의 인스턴스에 속해있는 함수이다. **인스턴스(instance)**는 클래스에 값이 할당되어 서로 구분되는 특수한 실제 상태이다.

7.2 Define Class

클래스 정의(Class definition)은 클래스의 속성과 메소드를 정의하는 과정이다.

```
class MyClass():
    pass
```

7.3 Instance

클래스의 인스턴스는 **init**() 메소드에 의해 정의된다. 클래스의 모든 메소드는 self를 첫번째 매개변수로 입력받는데, self는 인스턴스의 그 자체로 객체 자기 자신을 참조하는 argument가 된다 .

```
class MyClass():
    def __init__(self, param_1):
        self.attribute_1 = param_1
mc_2 = MyClass("arg_1")
```

7.4 Method

클래스의 메소드는 클래스 내부에 함수 정의로 생성되며, 생성된 메소드는 인스턴스 뒤에 .[function]()을 통해 사용할 수 있다.

```
class MyClass():
    def __init__(self, param_1):
        self.attribute_1 = param_1
    def add_20(self):
        self.attribute_1 += 20
mc_3 = MyClass(10) # mc_3.attribute is 10
mc_3.add_20() # mc_3.attribute is 30
```

7.5 list.append() method

```
class NewList(DQ) :             # 클래스를 정의한다
    def __init__(self, initial_state) :
        """
        __init__, initializer를 통해 input값을 객체의 attribute로 정의한다
        attribute는 variable, method 둘다 의미
        """
        self.data = initial_state

    def append(self, new_item) :        # 메소드를 정의한다
        # 객체의 attribute를 저장하였기 때문에 불러올 때도 객체.변수명을 통해 불러올 수 있다
        new_item_list = [new_item]
        self.data = self.data + new_item_list

my_list = NewList([1, 2, 3, 4, 5])
print(my_list.data)

my_list.append(6)
```

8. Cleaning and Preparing Data in Python

Python에서 CSV파일을 사용해서 작업할 때 대부분의 데이터의 형태는 "list of lists" 형태로 존재하게 된다. 이때 각각의 데이터는 문자열 데이터로 존재한다. 정돈되지 않은 데이터를 데이터 분석가를 위해 정돈하는 작업을 **데이터 정제(Data Cleansing)**이라고 한다.

8.1 replace substring

문자열의 일부분을 **substring**이라고 한다. 문자열 내부에서 특정 substring을 다른 substring으로 변경하려면 str 객체의 replace() 메소드를 사용할 수 있다.

```
for row in moma :
    nationality = row[2]
    nationality = nationality.replace('(', '')
    nationality = nationality.replace(')', '')
    row[2] = nationality
```

8.2 Convert uppercase

문자열의 첫 단어를 대문자로 변경하고 싶을 땐 title() 메소드를 사용한다.

```
for row in moma :
    nationality = row[2]

    nationality = nationality.title()

    if not nationality :
        nationality = "Nationality Unknown"

    row[2] = nationality
```

8.3 Check a string for the existence of a substring

딕셔너리에 어떤 값이 있는지 확인하기 위해 "in"을 사용했었다. 문자열도 마찬가지로 어떤 substring이 문자열 내부에 존재하는지 확인하려면 in을 사용해 불린 타입의 결과를 출력할 수 있다.

```
bad_chars = ['c.', 'C.', "'s"]

def strip_characters(string):
    for char in bad_chars:
        string = string.replace(char,"")
    return string
```

8.4 Split a string

문자열을 특수문자를 기준으로 리스트의 원소로 분리할 수 있다. split() 메소드는 문자열을 특수 문자를 기준으로 리스트를 생성하고, join() 메소드는 문자열을 특수문자를 기준으로 하나의 문자열로 합친다.

```
def process_data(year) :

    if '-' in year :
        year1, year2 = year.split('-')
        mean_year = round((int(year1) + int(year2))/2)

        return mean_year

    else :
        year = int(year)

        return year
```

8.5 String formatting

문자열 포맷팅은 외부 데이터를 문자열에 자동으로 입력할 때 사용하는 방법으로 '{}'안에 저장하고 싶은 데이터의 변수명을 적고 .format()뒤에 변수명을 입력한다. 포맷의 데이터 타입과 옵션을 지정하기 위해서 %뒤에 %s 혹은 %.2f와 같이 사용한다.

```
continents = "France is in {} and China is in {}".format("Europe", "Asia")
```

9. Date Times

9.1 Importing packages

datetime 패키지는 시계열 형식의 데이터를 효율적으로 다룰 수 있는 속성과 메소드를 제공한다. 패키지를 선언하는 방법은 다음과 같다.

```
import datetime as dt
```

9.2 datetime.datetime

datetime 패키지의 datetime 클래스는 day, month, year 등의 속성을 가지고 있다. 또한 strptime(), strftime() 등의 메소드를 가지고 있다. strptime() 메소드는 문자열의 데이터를 지정된 형식의 datetime 객체로 변환해주고, strftime() 메소드는 datetime 객체의 데이터를 문자열 데이터로 변환한다.

```
dt_object = dt.datetime(1984, 12, 24)           # __init__ constructor, attribute를 지정한다
dt_string = dt_object.strftime("%d/%m/%Y")

# 월별 방문객 frequency table

visitors_per_month = {}

for row in potus :
    appt_start_date = row[2]                # dt.datetime instance를 저장

    month_year = appt_start_date.strftime("%B, %Y") # strftime() method 적용

    if month_year in visitors_per_month :
        visitors_per_month[month_year] += 1
    else :
        visitors_per_month[month_year] = 1
```

9.3 datetime.timedelta

timedelta 패키지는 datetime 클래스 객체간 시간의 변화에 대한 정보를 가지고 있는 객체로 datetime - datetime 혹은 datetime + datetime 연산을 진행할 때 생성되고, datetime 객체 뒤에 시간의 변화를 줄 때 사용된다.

```
d1 = dt.date(1963, 2, 26)
d1_plus_1wk = d1 + dt.timedelta(weeks=1)
```

10. Binary and Positional Number Systems

컴퓨터는 숫자를 인간이 인식하는 방법과는 다르게 인식한다. 컴퓨터는 전기회로가 꺼지고 켜지는 이진법 체계속에서 작동한다. 모든 숫자는 2진법 체계속에서 표현된다. 2진법으로 표현되는 각 자리수는 **비트(bit)**라고 불리며 8비트는 **1 바이트(Byte)**라고 불린다. 8진법과 16진법은 이진법으로 표기된 자료를 인간이 읽기쉽게 표현한 형태이다. 16진법은 주로 색깔을 표현하는데 사용되며 8진법은 파일시스템의 권한을 설명하는데 사용된다.

Python의 int() 함수에는 두번째 parameter에 진법 arguments를 입력하게 되면 첫번째 입력된 숫자를 해당 진법으로 인식하여 십진법으로 변환하게 된다. bin() 함수는 10진법의 수를 2진법으로 표기한다.

```
# Converting a string representing a number of given base to base 10
int('1101', 2)

# Converting from base 10 to binary
bin(1234)

# Converting from base 10 to octal
oct(1234)

# Converting from base 10 to hexadecimal
hex(1234)
```

11. Encoding and Representing Text

11.1 Encoding

인코딩(Encoding) 정보의 형태나 형식을 변환하는 처리나 처리방식으로 문자 인코딩의 경우 문자들의 집합을 부호화하는 방법이다. 컴퓨터는 0과 1이외의 숫자는 입력받지 않기 때문에 문자를 표현하기 위해선 인코딩이 필요하다. **ASCII** 인코딩은 128개의 문자코드를 가지고 있다. ASCII 인코딩의 경우 문자이외에도 컴퓨터를 제어하기 위한 제어문자들도 포함되어 있다.

인코딩 된 문자열은 저장소 내부에 저장이 가능하며 1bytes로 저장되어 있어 bytes 객체로 정의되어 있다. 인코딩된 문자열은 다른 인코딩 방식으로 전환할때 깨진 문자열을 출력하지만 별다른 처리가 필요없다. 그렇기 때문에 해당 파일이나 문서가 어떤 방식으로 인코딩 되어 있는지 알고 디코딩 하는 작업이 매우 중요하다.

```
'Data Quest'.encode(encoding='ascii')
```

11.2 Representing Text

알파벳을 제외한 문자열은 1바이트로 표현하기엔 제한이 있어 다른 인코딩 방식이 필요하다. 이때 등장한 방식들이 CP949, EUC-KR, UTF-8, UTF-16이다. 해당 방식을 **유니코드(Unicode)**라고 하며 키와 값이 1:1로 매핑된 형태의 코드를 의미한다.

11.3 Check encoding and read files

```
import chardet

with open("kyto_restaurants.csv", mode = "rb") as file :
    raw_bytes = file.read()
    detected_encoding = chardet.detect(raw_bytes)['encoding']

import csv

with open("kyto_restaurant.csv", mode = 'r', encoding = detected_encoding) as file :
    rows = list(csv.reader(file))
```

12. Reading and Writing Files

12.1 Open file with a context manager

파일을 오픈하면 반드시 닫는 작업을 수행해야 한다. **Context manager**은 파일을 열고 닫는 작업을 자동으로 수행해준다. 또한 파일에 데이터를 작성하거나 추가하는 작업을 할 수 있으며, 인코딩 방식을 지정해 파일을 열 수 있다.

```
with open('data.txt') as file:
    for line in file:
        print(line)
```

12.2 Read file with encoding

```
with open('data.txt', mode = 'r', encoding = 'UTF-8') as file :
    ...
```

12.3 Write to a file

```
with open('data.txt', mode='w') as file:
    file.write('line 1\n')
    file.write('line 2\n')
```

12.4 Append to a file

```
with open('data.txt', mode='a') as file:
    file.write('line 3\n')
    file.write('line 4\n')
```

12.5 Convert a CSV file from one encoding to another

```
import csv
with open('original.csv', encoding=original_enc) as file:
    rows = list(csv.reader(file))

with open('new.csv', mode='w', encoding=new_enc) as file:
    writer = csv.writer(file)
    for row in rows:
        writer.writerow(row)
```

13. Memory and Disk Storage

13.1 Two's complement representation

컴퓨터는 음수를 표현하기 위해 **2의 보수(two's complement representation)**을 사용한다. 예를들어 8bit의 경우 가장 높은 자릿수가 음수로 표현되어 -127 ~ 126의 수를 표현 가능한 방법이다. 8bit의 경우 0111111 에서 1000000으로 자리수가 올라갈 때 음수로 바뀌게 된다. 이를 **overflow**라고 표현한다.

주로 int64, int32와 같이 고정 길이 방식을 사용하여 음수를 표현하며 np.binary_repre() 메소드를 통해 2진법의 수를 확인할 수 있다.

```
np.binary_repr(2147483647, width=32)
```

13.2 Disk usage

ASCII 인코딩 문자의 경우 1Bytes의 크기를 갖고 있지만 유니코드의 경우 2Bytes의 크기를 가지고 있다. 또한 각 인코딩은 인코딩이 메모리 상에 기본적으로 차지하고 있는 **overhead**가 존재한다. os.path.getsize(), sys.getsizeof() 메소드를 통해 데이터의 크기를 확인할 수 있다.

```
import sys
sys.getsizeof([0, 4, 2])

import os
os.path.getsize('filename')
```