<https://wikidocs.net/43>

1주차 : chap. 1 – 4

* 인터프리터 & 컴파일러  
  인터프리터 = 동시 통역기. 결과를 즉각 제공  
  컴파일러 = 입력을 모두 받고 한 번에 기계어로 바꾸어 줌.
* 코드 실행
  + IDLE  
    IDLE를 통해 코드 작성 및 실행을 할 수 있음
  + 윈도우 명령프롬프트  
    <윈도우키> + <R> -> 실행 창(<열기> 우측에 입력하는 공간)에서 cmd 입력
    - where python  
      파이썬 실행 파일이 어디 있는지 확인할 수 있음.
    - cd 파일 경로  
      해당 파일로 이동
    - dir  
      현재 디렉토리(폴더)의 파일 목록을 모두 출력
    - dir 파일명\*  
      해당 파일이 있는지 확인
    - python 경로(\blabla\blabla\file.py)  
      파이썬 인터프리터로 이 프로그램 전체를 실행
* 연산자  
  연산은 일반적으로 왼쪽 항에서부터 수행되고, 괄호를 사용하는 경우 순서를 임의로 바꾸어 줄 수 있음. 이때, 조건문에 대해 A and B일 때, A가 False임을 확인하면 B를 고려하지 않고 바로 False를 반환함. 즉, 다음 수행할 연산이 필요 없는 경우에는 다음 수행할 연산을 수행치 않음.
  + 산술계산 연산자 및 함수  
    +: 덧셈  
    -: 뺄셈  
    \*: 곱셈  
    /: 나눗셈  
    //: 소수점 이하 생략  
    %: 나머지  
    x \*\* y : x의 y제곱  
    divmod(x, y): 결과로 (a, b). x/y의 계산에서 a는 몫, b는 나머지.  
    round(x, y): x를 소수점 y번째 자리로 반올림. 이때, round(2.675, 2)의 수행은 부동소수점 연산의 한계로 2.67이 출력됨.[[1]](#footnote-1)  
    sqrt(x) : x의 제곱근을 구함. from math import \*(또는 sqrt) 를 한 후 사용 가능.
  + 논리 연산자  
    x == y : x와 y가 같은 지 검사하여 True, False 값을 반환  
    x != y : x와 y는 다른 지 검사하여 True, False 값을 반환  
    x and y : x와 y에 대해서 and 연산. x와 y가 모두 True여야 True를 반환.  
    x or y : x와 y에 대해서 or 연산. x와 y 중 하나라도 True이면 True 반환.
* 변수  
  변수 이름 = 값  
  -> 값을 변수에 대입한다.  
  -> 값 : 숫자(수 입력 시), 문자열(따옴표로 묶은 경우)
  + 문자열에 사용되는 키워드  
    문자열1 in 문자열2 : 문자열1이 문자열2 안에 있다면 True, 없다면 False를 반환.
  + 변수끼리 사칙연산 가능  
    변수에 문자열이 담긴 경우 : 문자열 이어붙임.
  + 복합대입연산자  
    ‘사칙연산=’ (예. +=, -=, \*=)  
    위와 같은 형태
* 자료형  
  컴퓨터는 모든 정보를 이진수로 처리. 해당 이진수에 대해 어떠한 값으로 읽을지에 대한 것이 자료형으로 결정됨.  
  type(값) – 값의 자료형을 확인할 수 있음.
  + 숫자
    - 정수(int)
    - 부동소수점수(float)  
      단순히 소수점 이하를 표현할 수 있는 수라고 생각하면 됨.
    - 복소수(complex)
    - 숫자 -> 문자열  
      str(숫자)  
      -> 숫자 123으로 문자열 ‘123’을 얻게 됨
  + 시퀀스(sequence)
    - 문자열(str)
      * 문자열의 글자 바꾸기  
        문자열 = ‘문자’ + 문자열[범위] -> 이와 같이 오려 붙이는 방식으로만 문자를 수정할 수 있음.  
        문자열[숫자] = ‘문자’ -> 올바르지 않은 명령어. 문자열에 들어있는 글자는 대입으로 바꿀 수 없음.
      * 문자열[a:b]  
        - 문자열에서 인덱스 a부터 b-1까지의 문자를 복사할 수 있음.  
        - 시작 인덱스(a)가 0일 때는 생략할 수도 있음.  
        - 인덱스 0에서 더 내려가, 가장 마지막 문자는 -1, 그 전 문자는 -2, …과 같이 인덱스를 부여할 수 있다. 복사하는 원리는 기본적으로 같다.  
        - a와 b를 모두 생략하면 문자열 전부를 복사한다.  
        - 문자열[::-1] 과 같이 쓰면 역순으로 전체를 복사한다.
      * lower()  
        문자열.lower()  
        문자열의 문자를 모두 소문자로 바꾼 문자열을 얻을 수 있음.
      * replace()  
        문자열.replace(a, b)  
        문자열의 문자 a를 문자 b로 바꾼 문자열을 얻을 수 있다
      * find()  
        문자열.find(‘문자’)  
        문자열에서 문자가 몇 번째 자리에 있는지를 반환. 만약 존재하지 않는다면 -1을 반환.
      * strip()  
        문자열.strip(‘문자’= 양쪽 끝 공백)  
        -> 문자열에서 ‘문자’를 제거할 수 있음. 이때, 디폴트 값은 양쪽 끝 공백임.  
        문자열.rstrip(‘문자’= 우측 끝 공백) / 문자열.lstrip(‘문자’= 좌측 끝 공백)  
        -> 문자열의 오른쪽 / 왼쪽에서 ‘문자’를 제거할 수 있음.
      * split()  
        문자열.split(‘문자’)  
        -> 문자열에서 ‘문자’를 구분자 삼아 분할하고 이를 담은 리스트를 반환.
      * splitlines()  
        문자열.splitlines()  
        -> 줄바꿈을 기준으로 문자열을 분할하여 이를 담은 리스트를 반환
      * 문자열 -> 숫자  
        int(문자열), float(문자열)
      * 문자열 -> 리스트  
        1. for문 이용  
        for 변수 in 문자열:  
         빈 리스트.append(변수)  
        2. list()  
        list(문자열)  
        -> 문자열의 문자 하나 하나를 원소로 만들어 갖는 리스트로 변환.
    - 리스트(list)  
      리스트 이름 = [원소1, 원소2, …]
      * 리스트의 원소에 새로운 값 지정  
        리스트[인덱스] = 값
      * 리스트의 원소로는 리스트도 가능함.  
        ex. orders = [‘potato’, [‘pizza’, Coke’, ‘Salad’], ‘hamburger’]
      * 리스트를 사용해서 행렬을 표현할 수도 있음.(2차원 리스트)  
        ex. matrix = [[1, 2, 3], [4, 5, 6], [7, 8, 9]]
      * 리스트 이름[인덱스]  
        인덱스에 해당하는 리스트의 원소 반환
      * map()  
        map(함수, 리스트)  
        리스트로부터 원소를 하나씩 꺼내서 함수를 적용시킨 다음. 그 결과를 새로운 리스트에 담음. 해당 함수의 반환값의 자료형은 map. 리스트의 형태로 바꾸고 싶다면 list(map(함수, 리스트))를 이용하여 결괏값을 리스트로 받을 수 있음.
      * len()  
        len(리스트 이름)  
        리스트의 원소 개수 반환
      * remove()  
        리스트명.remove(원소)  
        리스트에서 원소를 제거
      * filter()  
        filter(함수, 리스트)  
        리스트의 원소들을 함수에 적용시켜서 결과가 참인 값들로 새로운 리스트를 생성. 반환값의 자료형은 filter이므로 list()를 적용시켜 결과를 리스트로 받을 수 있음.
      * append()  
        리스트.append(원소)  
        -> 리스트 끝에 원소를 추가
      * sort()  
        리스트.sort()  
        -> 원소를 크기 순으로 정렬
      * insert()  
        리스트.insert(인덱스, 원소)  
        -> 리스트의 인덱스 위치에 원소를 삽입.
      * del  
        del 리스트[인덱스]  
        리스트의 인덱스 위치의 원소를 삭제.
      * pop()  
        리스트.pop(인덱스)  
        리스트에서 인덱스 위치의 원소를 빼내어 반환. 기존 리스트에서는 해당 원소가 삭제되고 그 원소를 반환하는 것.
      * sum()  
        sum(리스트)  
        리스트 원소들의 총합을 구할 수 있음.
      * 리스트 -> 튜플  
        튜플명 = tuple(리스트명)  
        리스트 ‘리스트명’으로 튜플 ‘튜플명’을 만듦.
      * 리스트 -> 세트  
        set(리스트명)
    - 튜플(tuple)  
      튜플명 = (원소 1, 원소 2, …)  
      튜플을 만들 때 위와 같이 괄호를 써도 되고 안 써도 됨. 다만 원소가 없는 튜플을 만들 때는 괄호를 꼭 써야 함.(ex. empty = ()) 원소가 하나만 있는 튜플을 만들 때는 원소 뒤에 콤마(,)를 꼭 찍어야 함.(ex. one = 5,)
      * 튜플은 리스트와 달리 원소값을 직접 바꿀 수 없음. 때문에, 문자열에서 했던 것처럼 오려 붙이는 방법 써야 함.  
        튜플명[a:b]의 형식 이용 가능.
      * 튜플 -> 리스트  
        리스트명 = list(튜플명)  
        튜플 ‘튜플명’으로 리스트를 만듦.
    - 사용자 정의 클래스
  + 딕셔너리(dict)  
    {key : value}  
    딕셔너리는 위의 형식으로 구성되며, 키(key)와 값(value)의 짝으로 구성된다. 이를 매핑(mapping)이라 한다.  
    문자열, 리스트, 튜플은 숫자로 된 인덱스를 사용하는 반면 딕셔너리는 키를 이용한다는 점에서 큰 차이를 보임.   
    해싱 기법을 이용하기에 자료가 순서대로 저장되지 않음.[[2]](#footnote-2)
    - 빈 딕셔너리 선언  
      딕셔너리명 = {}
    - 딕셔너리 선언  
      딕셔너리명 = {키:값, 키:값, …}
    - 원소 추가  
      딕셔너리명[키] = 값
    - 원소 삭제  
      del 딕셔너리명[키]
    - 딕셔너리명[키]  
      이에 해당하는 값을 반환
    - 딕셔너리 -> 리스트  
      딕셔너리명.keys()  
      -> 딕셔너리의 키들을 새로운 리스트에 담는다.  
      딕셔너리명.values()  
      -> 딕셔너리의 값들을 새로운 리스트에 담는다.
    - 검색  
      키 in 딕셔너리  
      -> 키가 딕셔너리에 있다면 True, 없다면 False를 반환.
  + 불(bool)  
    참, 거짓을 표현. True, False가 이에 해당.
  + 세트(set)  
    이름 = {값1, 값2, …}  
    원소의 순서가 유지되지 않고, 중복 원소를 갖지 않는 ‘집합’으로서의 특징을 지님. 집합 연산을 사용할 수 있음.
    - 세트 선언  
      세트명 = set()  
      -> 빈 세트 선언  
      세트명 = {원소, 원소, …}  
      -> 원소가 있는 세트 선언
    - add()  
      세트명.add(원소)  
      -> 세트에 원소를 추가
    - 집합 연산
      * 교집합(&)  
        세트1 & 세트2  
        -> 세트1과 세트2의 교집합 결과를 반환

\*\*\* 세트를 리스트에 담은 후 set의 메서드를 쓸 수도 있음 \*\*\*  
set.intersection(\*리스트)  
-> 리스트의 원소로 들어간 세트에 대해 교집합.

* + - * 합집합(|)  
        세트1 | 세트2  
        -> 세트1과 세트2의 합집합 결과를 반환  
        \*\*\* 세트를 리스트에 담은 후 set의 메서드를 쓸 수도 있음 \*\*\*  
        set.union(리스트)  
        -> 리스트의 원소로 들어간 세트에 대해 합집합.
      * 차집합(-)  
        세트1 – 세트2  
        -> 세트1에서 세트2를 빼준 결과를 반환.
  + 시퀀스 함수
    - reduce()  
      reduce(함수, 시퀀스, 초깃값=0)  
      시퀀스(문자열, 리스트, 튜플)의 원소들에 대해 누적해서 연산하는 것.  
      reduce(lambda x, y : x + y, range(5))의 경우, 0과 0의 합을 x로, 1을 y로 하여 x+y를 수행하고 결과로 나온 1을 다시 x에 넣는다. 이후, 2를 y로 하여 x+y를 수행하여 결과로 나온 3을 다시 x에 넣는다. 이와 같은 수행을 리스트의 마지막 값에까지 적용하여 나온 결과를 반환한다.
* 제어구조
  + 반복문
    - while  
      while 조건:  
       body  
      -> 조건을 검사하여 만족되는 동안 body를 반복. body 수행의 결과로 조건을 만족하지 못하게 되면 루프를 빠져 나옴.
    - for  
      for x in y:  
       body  
      -> y 범위 안에서 각 항목 x에 대하여 body를 수행.  
      -> range()와 함께 자주 사용됨.
      * range(a, b)  
        a 이상 b 미만의 수가 리스트의 형태로 반환.
  + 조건문
    - if – elif – else  
      if 조건 1:  
       body 1  
      elif 조건 2:  
       body 2  
      else:  
       body 3  
      -> 조건 1을 만족하면 body 1을, 이를 만족하지 않고 조건 2를 만족하면 body 2를, 조건 1, 조건 2 모두 만족하지 않으면 else 문에 있는 body 3를 수행. 이때, elif는 여러 개가 사용될 수 있다.
  + match-case  
    match 식:  
     case 0:  
     body0  
     case 1:  
     body1  
     case 2:  
     body2  
    …  
    -> 식의 결과를 ‘case 숫자’의 숫자와 비교하여, 맞는 케이스에 대한 body를 수행.  
    -> 식에서는 두 개의 값을 평가할 수 있음(튜플). 이 경우 각 case들은 두 개의 값을 평가 대상으로 삼음. 이때, 아무 값이나 상관없다는 뜻의 ‘\_’를 많이 사용하게 됨. case \_:는 “그 밖의 모든 경우”라는 의미임.
* 함수  
  def 함수명(매개변수명):  
   body  
  -> 매개변수는 필요에 따라 없게 할 수도 있음.  
  -> body에 return 문이 있을 수 있음. return x의 경우, 함수가 x를 반환함. return 문이 없다면 함수는 아무 값도 반환하지 않음(None).  
  -> 함수를 정의할 때, 매개변수에 \*를 붙여두면 그 이후에 들어오는 것은 모두 튜플에 집어 넣음. ex. 함수명(x, y, \*rest)
  + 지역변수  
    함수 안에서 만들어진 변수. 함수가 호출되면 만들어져서, 함수의 실행이 끝날 때 함께 소멸됨.  
    global 변수명: 어떤 변수를 전역변수(global)로 사용. 이를 통해 함수 안에서 전역변수를 만들 수 있음. 그 경우, 함수의 실행이 끝난 다음에도 없어지지 않음.
  + 전역변수  
    함수 밖에서 만들어진 변수. 함수의 생성과 소멸과 관계없이 선언된 변수. 함수도 이를 이용할 수 있음. 프로그램이 복잡해질수록 전역변수와 관련하여 고려해야할 사항이 많아지므로 무조건 전역변수가 좋은 것은 아님.
  + 람다(lambda)  
    함수를 한 줄로 표현할 수 있도록 하는 형식.  
    1. (lambda 매개변수 : 반환값으로 사용할 식)(매개변수에 들어갈 값)  
    2. lambda 매개변수 : 반환값으로 사용할 식 -> 이것 자체가 함수  
    다음과 같이 이용할 수 있음.  
    함수명 = lambda 매개변수 : 반환값으로 사용할 식  
    함수명(값)  
    -> 값에 대한 함수 실행 결과 발생.
* 유용한 함수
  + input()  
    input(문자열)  
    문자열을 출력하고 입력값을 받음.
  + print()  
    print(내용)  
    괄호 안의 내용을 출력.  
    a와 b가 숫자이고, 이를 문자열에 함께 쓰고 싶을 때, 다음과 같이 쓴다.  
    print(f’{a} 문자열 {b}’) 혹은 print(f’{a} 문자열 {b:자릿수d}’)
  + bin()  
    bin(십진수)  
    십진수를 이진수로 나타낼 수 있음. 이때, 0b가 앞에 붙어서 나오는데 이는 이진수라는 의미의 표현임.
  + ord()  
    ord(문자)  
    문자에 해당하는 ASCII 코드값을 반환
  + chr()  
    chr(ASCII 코드값)  
    ASCII 코드에 해당하는 문자를 반환.

1. 소수를 기약분수로 표현했을 때, 분모가 2의 거듭제곱 꼴이 아니라면, 부동소수점으로 표현했을 경우 오차가 발생함. [↑](#footnote-ref-1)
2. 순서대로 원소를 검색하는 것이 아닌, 키를 바탕으로 원소를 검색하는 기법 [↑](#footnote-ref-2)