**Python [Hybrid Languages]**

**Abstraction**

파이썬(Python)은 1980년대 말 고안되어 네덜란드 CWI의 귀도 반 로섬(Guido van Rossum)이1989년 12월 구현하기 시작하였다. 이는 역시 SETL에서 영감을 받은 ABC 언어의 후계로서, 예외 처리가 가능하고, 아메바 OS와 연동이 가능하였다. 1991년 귀도 반 로섬이 발표한 첫 번째로 발표한 파이썬은 고급 프로그래밍 언어로, 플랫폼 독립적이고, 인터프리터(Interpreter)형식, 객체지향적(Object-Oriented), 동적 타이핑(dynamically typed) 대화형 언어이다. 파이썬이라는 이름은 귀도가 좋아하는 코미디 프로그램 <Monty Python’s Flying Circus>에서 따서 지은 것이다. 파이썬은 비영리의 파이썬 소프트웨어 재단이 관리하는 개방형, 공동체 기반의 개발 모델을 가지고 있다. 그리고 2000년 10월 16일 배포된 파이썬 2.0에서는 많은 기능이 추가되었다. 그중 전면적인 쓰레기 수집기(GC, Garbage Collector) 탑재와 유니코드 지원이 특징적이다. 그러나 가장 중요한 변화는 개발 절차 그 자체로, 더 투명하고 공동체 지원을 받는 형태가 되었다. 파이썬3000(혹은 파이썬3k)이라는 코드명을 지닌 파이썬의 3.0 버전의 최정판이 긴 테스트를 거쳐 2008년 12월 3일자로 발표되었다. 2.x대 버전의 파이썬과 하위호환성이 없다는 점이 가장 큰 특징이다. 파이썬 3의 주요 기능 다수가 이전 버전과 호환되도록 2.6과 2.7버전에도 반영되기도 했다. 2.x대 버전 과의 차이를 간략하게 요약하면 사전형과 문자열형과 같은 내장 자료형의 내부적인 변화 및 일부 구형의 구성 요소 제거, 표준 라이브러리 재배치, 향상된 유니코드 지원, 한글 변수 사용 가능, print 명령문이 print() 함수로 바뀌게 된 점 정도이다.

파이썬은 다양한 프로그래밍 패러다임을 지원하는 언어이다. 객체 지향 프로그래밍과 구조적 프로그래밍을 완벽하게 지원하며, 함수형 프로그래밍, 관점 지향 프로그래밍 등도 주요 기능에서 지원된다. 이러한 점에서 파이썬을 Hybrid Language라고 본다. 파이썬 만의 철학이 있는데 그 중 핵심적인 것은 다음과 같다. “아름다운 것이 추한 것보다 낫다.”, “명시적인 것이 암시적인 것 보다 낫다.”,”단순함이 복잡함보다 낫다.”, “복잡함이 난해한 것보다 낫다.”, “가독성은 중요하다.” 등이다. 파이썬은 언어의 핵심에 모든 기능을 넣는 대신, 사용자가 언제나 필요로 하는 최소한의 기능만을 사용하면서 확장해나갈 수 있도록 디자인되었다. 문제를 해결하는 방법이 단 한 가지가 아니라고 생각하는 프로그래밍 언어와는 대조되게, 파이썬은 다른 사용자가 썼더라도 동일한 일을 하는 프로그램은 댗로 모두 비슷한 코드로 수렴한다. 라이브러리는 기본 기능에 없는 많은 기능을 제공한다.

개인적으로 생각하는 파이썬이 가지는 가장 큰 장점은 초보자들이 쉽게 배우고 사용할 수 있고, 전문가들에게도 또한 강력한 프로그램이 솔루션을 제공한다는 점이다. 프로그래밍이라는 것은 인간이 생각하는 것을 컴퓨터에게 지시하는 행위라고 할 수 있다. 하지만 인간이 생각했던 것과 컴퓨터가 할 수 있는 것 사이의 간극과 미묘한 차이들이 프로그래밍을 배우는 것을 어렵게 만든다. 하지만 파이썬은 사람이 생각하는 방식을 그대로 표현할 수 있는 언어다. 따라서 프로그래머는 굳이 컴퓨터의 사고 체계에 맞추어서 프로그래밍을 하려고 애쓸 필요가 없다. 문법이 쉽고 직관적으로 이해할 수 있도록 되어 있다. 그리고 파이썬은 오픈 소스이므로 무료이다. 사용료 걱정 없이 언제 어디서든 파이썬을 다운로드 하여 사용할 수 있다. 또한 파이썬은 의도적으로 간결하게 만들어졌다. 마지막으로 파이썬의 문법에서 가장 잘 알려진 특징을 들여쓰기를 이용한 블록 구조를 들 수 있다. 이것은 보통 C 등에서 쓰이는 괄호를 이용한 블록 구조를 대신한 것으로 줄마다 처음 오는 공백으로 눈에 보이는 블록 구조가 논리적 제어 구조와 일치하게 하는 방식이다. 이는 사용자로 하여금 들여쓰기를 강제하여 코딩 스타일을 제한한다는 단점으로 볼 수도 있지만, 코드의 가독성 면에서 매우 우수하다. 이러한 점에서 파이썬은 초보자들이 배우기 쉽고, 잘 다루기 쉬운 언어라고 할 수 있다. 또한 파이썬은 다양한 플랫폼에서 사용될 수 있고, 라이브러리(모듈)가 풍부하여 이미 여러 교육 기관, 연구 기관 및 산업계에서 활용되고 있다. 또한 파이썬은 순수한 프로그래밍 언어로서의 기능 외에도 다른 언어로 쓰인 모듈들을 연결하는 풀언어(glue language)로써 자주 이용된다. 실제로 파이썬은 많은 응용 프로그램에서 스크립트 언어로 채용되고 있다. 파이썬의 주요 특징 중 하나인 동적 타이핑은 프로그래머로 하여금 유연한 사고를 프로그램에 더 잘 반영할 수 있다는 점에서 또한 힘을 발휘한다고 할 수 있겠다.

**Data Type**

자료형(Data Type)이란 ‘데이터를 표현하는 방식’이다. 프로그래밍 언어에서 실수, 정수, 불린 등등의 여러 종류의 데이터를 식별하는 분류로서, 해당 자료형에 대입할 수 있는 값, 수행할 수 있는 연산 및 명령 등을 결정하므로 매우 중요하다.

1. 숫자형

숫자형이란 숫자 형태로 이루어진 자료형을 말한다. 우리가 흔히 사용하는 10, 20과 같은 정수(int), 3.14, 95.25와 같은 실수(float)를 포함해 8진수와 16진수도 표현할 수 있다. 또한 허수부(j/J)가 포함된 복소수도 표현할 수 있다. 정수는 10진수/16진수/8진수/2진수로 표현할 수 있으며 실수형은 십진수 형식으로만 표현된다. 숫자에 소수점이나 지수를 표시하는 e/E가 있다면 실수형이고 그렇지 않다면 정수형으로 분류된다. 또한 허수부가 포함되어 있다면 복소수형이다. 0k, -0j, +0.0j, -0.0j도 내부적으로 복소수형으로 간주된다. 숫자 상수나 변수의 자료형을 확인하기 위해서는 내장 함수인 type()을 사용하면 된다. 정수의 범위에 제한이 없다는 점에서 다른 프로그래밍 언어와 차이점을 가진다. 이러한 점에서 파이썬이 매우 큰 숫자를 다뤄야 하는 분야나(천문학, 경제학 등) 데이터 분석에서 다른 언어보다 유리하다는 점을 짐작할 수 있다. 숫자형으로 할 수 있는 연산으로는 사칙연산 (+, -, \*, /) 이외에도 x의 y제곱을 나타내는 \*\* 연산자나 모듈러 연산을 뜻하는 % 연산이 있다. 또한 나눗셈 후 소수점 아랫자리를 버리는 // 연산자도 있다.

2. 문자열 자료형

문자열(String)이란 문자, 단어 등으로 구성된 문자들의 집합을 의미한다. 문자열은 큰 따옴표를 사용하여 만들 수 있다. “Life is too short, You need python”은 큰따옴표(“ ”)로 둘러싸여 있으므로 문자열이다. “123”에서 123은 비록 숫자이지만, 큰따옴표로 둘러싸여 있으므로 이 또한 문자열이다. 문자열을 만드는 방식으로는 큰따옴표를 사용하는 것 이외에도 3가지 방식이 더 있다. 작은따옴표(‘ ’)로 양쪽 둘러싸기, 큰따옴표 3개를 연속으로(“““ ”””) 사용하여 양쪽 둘러싸기, 작은따옴표 3개를 연속으로(‘‘‘ ’’’)사용하여 양쪽 둘러싸기까지 총 4개의 방식으로 문자열을 표현할 수 있다. 단순함이 자랑인 파이썬이 문자열을 만드는 방법을 4가지나 만들게 된 이유는 따로 있다. 문자열 안에 작은따옴표나 큰따옴표를 포함시키기 위해서, 여러 줄인 문자열을 변수에 대입하고 싶을 때 사용하기 위해서이다. 문자열에서는 여러 가지 연산이 가능하다. 문자열 곱하기는 문자열을 곱한 만큼 반복하라는 의미이다. a = “123”, a = a\*2 이면 a는 “123”을 두 번 반복하여 “123123”이 된다. 또한 문자열에서는 인덱싱(indexing)이 가능하고 이를 이용해 슬라이싱(slicing)이 가능하다. 문자열에서 각 문자는 0부터 시작하여 1씩 증가하는 인덱스를 가지게 된다. 예를 들어 a = “Life”라고 하면 a[0] = ‘L’, a[3] = ‘e’가 된다. 그렇다면 문자열에서 단순히 한 문자만을 뽑아내는 것이 아니라 “Life is good”이라는 문자열에서 ‘Life’나 ‘good’과 같은 단어를 뽑아내는 방법에 대해 궁금할 수 있다. 이것을 간편하게 해주는 것이 슬라이싱이다. 슬라이싱을 이용하여 a = “Life is good”에서 ‘Life’의 시작 인덱스가 0, 끝 인덱스가 3인점을 이용하여 b = a[0:4]라고 하면 b는 ‘Life’가 된다. 즉, 시작점에서 시작하여 끝점은 포함하지 않게 문자열을 잘라내는 것을 말한다. 이 이외에도 문자열에서 split, find, index등의 간편한 미리 정의된 함수들을 이용하여 문자열을 이용할 수 있다. 문자열은 immutable 자료형이라서 선언한 후에는 수정이 절대로 불가능하다. 문자열들을 수정해주는 많은 함수들이 있지만 이는 수정된 문자열을 원래 문자열에 복사하는 방식이지 실제로 문자열 원본이 수정되는 것은 아니다.

3. 리스트(list) 자료형

리스트를 이용하면 1, 3, 5, 7, 9라는 숫자 모음을 다음과 같이 간단하게 표현할 수 있다.

listA = [1, 3, 5, 7, 9]

리스트를 만들 때는 위와 같이 대괄호([ ])로 감싸 주고 각 요소값들을 쉼표(,)로 구분해주면 된다.

여러 가지 리스트들의 예시를 보도록 하자.

a = [ ]

b = [1, 2, 3]

c = [‘Life’, ‘is’, ‘good’]

d = [1, 2, ‘Life’]

e = [1, 2, [1, 3, 5]]

리스트는 a처럼 아무것도 포함하지 않는, 비어 있는 리스트일 수도 있고, b나 c처럼 숫자나 문자열을 요소값으로 가질 수도 있고, d처럼 서로 다른 자료형을 같이 요소값으로 가질 수도 있으며 e처럼 리스트 자체를 요소값으로 가질 수도 있다. 즉, 리스트 안에는 어떠한 자료형도 포함 시킬 수 있다. 리스트도 문자열처럼 인덱싱과 슬라이싱이 가능하다. 문자열과 마찬가지로 0에서 시작하여 1씩 증가한다. 리스트에서는 음수를 인덱스로 가질 수 있는데 마지막 요소가 -1이고, 앞으로 갈수록 1씩 감소한다. a = [1, 2, 3]에서 a[-1]은 3, a[-2]는 2가 되는 것이다. 리스트는 더하기, 곱하기와 같은 연산으로 두 개 이상의 리스트를 이어 붙이거나, 반복할 수 있다. 리스트는 문자열과 다르게 mutable하여 수정이 가능하다. 인덱스를 이용해 값을 직접 수정할 수 있고, 슬라이싱을 이용해 연속된 범위의 값을 수정할 수도 있다. 또한 append, del등의 여러 함수를 이용해 리스트를 이용, 수정할 수 있다.

4. 튜플(tuple) 자료형

튜플은 몇 가지 점을 제외하고는 리스트와 흡사하며 리스트와 다른 점은 다음과 같다. 리스티는 [ ]으로 둘러싸지만 튜플은 ( )으로 둘러싼다는 점과 리스트는 그 값의 생성, 삭제, 수정이 가능하지만 튜플은 immutable하여 그 값을 바꿀 수 없다. 또한 리스트와 유사하므로 인덱싱과 슬라이싱, 더하기와 곱하기가 전부 가능하다.

5. 딕셔너리(dictionary) 자료형

사람을 구분할 때 “이름” = “장태영”, “생일” = “950412” 등으로 구분할 수 있다. 파이썬은 이러한 대응 관계를 나타낼 수 있는 자료형을 가지고 있다. 다른 언어에서는 이를 해시(Hash)또는 연관 배열(Associative array)이라고 한다. 파이썬에서는 이러한 자료형을 딕셔너리라고 하는데, 단어 그대로 해석하면 사전이라는 뜻이다. 즉, soccer라는 단어에 “축구”라는 뜻이 대응되듯이 딕셔너리는 Key와 Value라는 것을 한 쌍으로 갖는 자료형이다. Key가 “Soccer”라면 Value는 “축구”가 되는 것이다. 딕셔너리는 리스트나 튜플처럼 순차적으로 해당 요소값을 구하지 않고 Key를 통해 Value를 얻는다는 점이 가장 차별적인 특징이다. 딕셔너리의 기본적인 구조는 다음과 같다.

A = {Key1 : Value1, Key2 : Value2, Key3 : Value3, …}

Key와 Value의 쌍 여러 개가 { }로 둘러싸여 있다. 각각의 요소는 Key : Value 형태로 이루어져 있고 쉼표(,) 로 구분되어 있다.

A = {‘name’ : ‘jang’, ‘id’ = ‘2014312541’, ‘hobby’ = ‘soccer’} 라는 딕셔너리가 있다고 해보자.

A[‘name’]이라고 하면 Key인 ‘name’에 대응되는 ‘jang’를 얻을 수 있다. 또한 딕셔너리는 정보를 추가하거나 삭제할 수 있다. 중요한 것은 이렇게 정보가 추가되고, 삭제되는 것에서 순서는 중요하지 않다는 점이다.

6. 집합(set) 자료형

집합은 집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형으로 set 키워드를 이용하여 만들 수 있다.

s = set(“Hello”) 라고 하면 s는 {‘e’, ‘H’, ‘l’, ‘o’}가 된다. 이런 결과가 나타나는 이유는 집합은 중복을 허용하지 않고, 순서가 없다는 특징이 있기 때문이다. 따라서 집합에서는 인덱싱과 슬라이싱이 불가능하다. 만약 두 기능을 이용하고 싶다면 집합을 리스트나 튜플로 변환한 후에 사용해야 한다. 집합에서는 & 를 이용하여 교집합, | 를 이용하여 합집합, - 를 이용하여 손쉽게 차집합을 구할 수 있다. 또한, add, update, remove등의 함수를 이용하여 값을 추가하거나 삭제할 수 있다.

7. 불(bool) 자료형

불 자료형이란 참(True)과 거짓(False)을 나타내는 자료형이다.

불 자료형은 참(True)과 거짓(False)만을 값으로 가질 수 있다.

True와 False는 파이썬의 예약어 이므로 항상 대소문자에 유의하여 사용해야 한다. 불에서 문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면 즉, “ “, [ ], ( ), { }등의 형태이면 거짓이 되고, 숫자형에서도 0이면 거짓으로 간주한다. 그 외의 모든 형태에 대해서는 참으로 간주한다.

8. 변수

파이썬에서 사용하는 변수는 모두 객체를 가리키는 것이라고도 할 수 있다. 객체란 자료형을 포함해 “파이썬에서 사용되는 모든 것”을 말한다. a = 3이라고 하면 3이라는 값을 가지는 정수 자료형(객체)이 자동으로 메모리에 생성된다. a는 변수의 이름이며, 3이라는 정수형 객체가 저장된 메모리 위치를 가리키게 된다. 즉, 3을 상수가 아닌 정수형 객체라고 보는 것이 이해가 쉬울 수 있다. 그렇다면 이렇게 메모리에 생성된 객체를 없앨 필요도 생긴다. 위의 예에서 3이라는 객체를 가리키는 a가 생기면 3의 레퍼런스 카운터(reference counter)가 1 증가하는데, 반대로 객체를 가리키는 것이 없어지면 1 감소한다. 이렇게 하여 레퍼런스 카운터가 0이되면 3이라는 객체는 자동으로 사라지게 된다. 이것이 파이썬에서 지원하는 쓰레기 수집(GC, Garbage Collection)이다. del함수를 이용하여 변수를 없앨 수 있지만 파이썬에서 자동으로 해주기 때문에 일일이 변수마다 del을 사용할 필요는 없다.

위에서 사용한 예시들처럼 a, b, c나 listA와 같은 것들이 모두 변수이다. 변수를 만들 때는 위의 예처럼 =(assignment) 기호를 사용한다. 파이썬에서는 다른 언어들처럼 자료형을 명시해줄 필요가 없다. 변수에 저장된 값을 스스로 판단하여 자료형을 알아내도록 한다.

**Sequence Control**

프로그래밍 언어에서 제어문은 순차적인 흐름을 조건에 따라 통제하는 역할을 한다. 사용자의 요구에 따라 프로그램이 동작하도록 하기 위해서이다. 제어문의 종류는 크게 두 가지인데 참과 거짓을 판별하는 조건문, 반복적인 작업을 할 때 사용하는 반복문이 있다.

1. 조건문

다음과 같은 상황을 가정해보도록 하자. ‘지갑이 있으면 밥을 사먹고, 지갑이 없으면 집에 돌아간다.’ 사람이라면 얼마든지 일어날 수 있는 상황을 가정하였다. 프로그래밍도 마찬가지로 사람이 만드는 것이기 때문에 조건에 따라 판단을 해야 하는 상황이 생길 수 있다. 이렇듯 프로그래밍에서 조건을 판단하여 해당 조건에 맞는 상황을 수행하는 데에 쓰이는 것이 바로 조건문이다. 조건문에는 if문이 있다.

앞서 말한 상황을 if문을 사용하여 Python에서 표현하자면 다음과 같다.

wallet = 1

if wallet :

print(“밥을 먹자!”)

else :

print(“집에 가자…”)

wallet에 입력된 1은 참을 뜻한다. 따라서 if문의 조건과 맞으므로 다음 문장이 수행되어 “밥을 먹자!”가 출력될 것이다.

if와 else의 기본적인 구조는 다음과 같다.

if 조건문 :

수행할 문장 1

수행할 문장 2

…

else if 조건문 :

수행할 문장 1

수행할 문장 2

…

else :

수행할 문장 1

수행할 문장 2

조건문을 테스트해서 참이면 if문 내에 있는 문장들이 수행된다. 만약 거짓이라면 순차적으로 다음의 else if 문이 수행된다. 이 경우에도 참이 되는 경우가 없다면 마지막의 else문 다음의 문장들이 수행된다. 따라서 else문은 if문 없이 독립적으로 사용될 수 없다는 점에 주의해야 한다. 또한 조건문을 만들 때에는 들여쓰기(indentation)에 주의해야 한다. 같은 if문 내에서는 들여쓰기를 하지 않으면 if문 내에 있는 문장이라고 인식하지 못하기 때문이다. 그리고 if 조건문 다음에는 콜론(:)을 붙여야 한다. 이는 else에서도 마찬가지로 적용된다.

2. 반복문

반복문은 조건을 판별하여 조건에 맞으면 해당 문장을 반복적으로 수행하도록 한다. 따라서 반복문 또한 if문처럼 조건문이 필요하며, 사용되는 반복문에 따라 조건에 맞게 반복해서 문장들이 수행되도록 한다. 반복문에는 for와 while이 있다.

- while

while은 반복해서 문장을 수행해야 하는 경우에 사용된다. while은 조건문이 참인 동안 while문 내에 포함된 문장들을 수행한다. while의 기본 구조는 다음과 같다.

while 조건문 :

수행할 문장 1

수행할 문장 2

…

while의 조건문의 조건이 맞는지 판별하고, 조건이 맞다면 문장들을 수행하고, 모두 수행되면 다시 while의 조건문의 조건이 맞는지 판별한다. 이렇게 하여 조건문이 맞지 않을 때까지 반복적으로 while문을 수행하게 된다.

예를 들어 다음과 같은 반복문을 실행한다고 하자.

i = 0

while i < 5 :

i += 1

print(i)

조건문이 i < 5 이므로 i가 5보다 작으면 while문 내의 문장들이 수행되는 것이다. i가 현재 0이므로 while내의 문장들이 수행된다. i += 1이므로 i가 1 증가하여 1이 되고, i를 출력한다. 이 과정을 반복하면 i가 4가 되고 이때 또 while이 수행되어 i가 5가 되고 i의 값인 5가 출력된다. 그리고 조건문을 비교하면 i가 5가 되었으므로 5보다 작지 않아서 더 이상 while문이 수행되지 않는다. 최종적인 결과로 1부터 5까지의 숫자가 출력된다.

반복문에서 break를 이용하면 반복문의 조건문이 참이라고 하더라도 강제로 반복문을 나가게 할 수 있다. 다음은 break를 이용하여 반복문을 강제로 빠져나가게 한 예시이다.

i = 0

while i < 5 :

i += 1

print(i)

if i == 3 :

break

위의 반복문에서 if를 이용하여 만약 i가 3이라면 break가 수행되도록 하였다. 따라서 i가 3이되면 break가 수행되어 while문을 빠져나갈 것이고, 그 이후에 출력될 4, 5는 출력되지 않고 1, 2, 3이 차례대로 출력될 것이다.

또한 반복문에서는 continue를 이용하면 반복문의 맨 처음 (조건문)으로 돌아가서 다시 조건이 참인지 판별할 수 있게 할 수 있다. 다음은 continue를 이용하여 홀수만을 출력하도록 하는 예시이다.

i = 0

while i < 10 :

i += 1

if i % 2 == 0 :

continue

print(i)

위의 반복문에서 if와 continue가 없다면 아마 1부터 10까지의 숫자가 출력될 것이다. 하지만 i가 2로 나누어 나머지가 0이면 짝수라는 뜻이므로 i가 짝수이면 continue가 수행되어 그 다음의 문장인 print가 수행되지 못하고 다시 조건문으로 돌아가게 된다. 따라서 10보다 작은 홀수인 1, 3, 5, 7, 9만 출력될 것이다.

- for

파이썬의 직관적이라는 특징을 가장 잘 대변해주는 것이 바로 반복문의 for문이다. for문은 매우 유용하고 문장 구조가 한눈에 쏙 들어온다는 장점이 있다. for문의 기본적인 구조는 다음과 같다.

for 변수 in 리스트(또는 튜플, 문자열) :

수행할 문장 1

수행할 문장 2

…

리스트나 튜플 또는 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 조건이 맞으면 다음 문장들이 수행된다. 실제로 for문이 어떻게 사용되는지 예시를 통해 알아보도록 하자.

listA = [1, 2, 3]

for i in listA :

print(i)

listA에는 1, 2, 3 총 3개의 숫자가 들어가 있고, for문을 해석해보면 조건문은 i에 리스트의 첫 번째 요소인 1부터 마지막 요소인 3까지 대입되고 다음 문장이 print(i)가 각각 수행된다는 것이다. 따라서 결과적으로 1, 2, 3이 출력될 것이다. 또한 for문은 range함수와 함께 사용할 수도 있다. range함수의 기본적인 구조는 range(시작점, 끝점, 변화)인데, 예를 들어 range(1, 10, 2)라고 하면 ‘1에서 시작하여 2씩 증가하여 10보다 작을 때까지’ 를 의미하고 시작점이 끝점보다 크고, 변화가 음수가 되도록 할 수도 있다. range(1, 10) 같은 형태가 되면 자동적으로 1씩 증가하도록 된다. 이를 for문에 적용하면 다음과 같이 동작한다.

for i in range(1, 11) :

print(i)

i가 range(1, 11)의 범위에서, 즉 1 <= i < 11에서 print(i)가 수행되는 것이다. 따라서 1부터 10까지 출력하고 i가 11이 되면 범위를 벗어나서 for문의 조건문을 만족하지 않아 더 이상 반복문을 수행하지 않게 된다.

**Function**

함수는 파이썬에서 작업의 기본 단위이다. 우선 함수를 쉽게 이해하기 위해 믹서기를 생각해 보도록 하자. 우리는 믹서기에 과일을 넣는다. 그리고 믹서기를 돌려 과일을 갈아 과일 주스가 나오도록 한다. 이 경우에 우리가 믹서기에 넣는 과일은 “입력”이 되고 과일 주스는 “출력(결과값)”이 된다. 그렇다면 함수가 바로 믹서기와 비슷하다. 함수는 입력 값을 전달해주면 그 입력 값을 가지고 작업을 수행하여 그 작업을 통해 얻어진 결과를 반환한다. 입력 값과 리턴 값이 없어도 함수가 될 수 있지만, 대부분의 함수는 입력 값에 대한 리턴 값이 존재하게 된다. 함수는 반복되는 작업을 줄이는데 효율적이다. 함수를 이용하는 또 다른 이유는 만든 프로그램을 함수화하면 프로그램의 흐름을 일목요연하게 볼 수 있다. 이렇게 되면 프로그램이 진행되는 흐름을 알기도 쉽고, 오류가 났을 때 어디에서 오류가 났는지도 쉽게 찾을 수 있다. 파이썬에서 함수는 print()처럼 미리 정의된 함수도 있고, 사용자가 직접 함수를 만들어 사용할 수 있는 사용자 정의 함수가 있다. 만약 함수를 직접 만들어 사용하고 싶다면 다음 기본 구조를 지키면 된다.

def 함수명(parameter 1, parameter 2) :

수행할 문장 1

수행할 문장 2

…

return 리턴 값

이것은 함수의 기본구조이지 모든 함수가 위와 같은 구조만을 가지는 것은 아니다. 예를 들면 print()와 같이 입력 값은 있지만 리턴 값은 없는 함수가 있을 수 있다. 자세한 내용은 다음의 매개변수에서 설명하도록 하겠다. 파이썬은 또한 스크립트 언어이기 때문에 함수 선언을 한 후에 그 함수를 호출할 수 있다. 그렇지 않으면 undefined error가 발생하게 된다.

그렇다면 매개변수에 대해 알아보도록 하자. 매개변수(parameter)와 인수(argument)는 혼용해서 사용되는 헷갈리는 용어이다. 매개변수는 함수에 입력으로 전달된 값을 받는 변수를 의미하고 인수는 함수를 호출할 때 전달하는 구체적인 입력 값을 의미한다. 일반적인 함수의 전형적인 예를 통해서 알아보도록 하자.

def sum(a, b) :

result = a + b

return result

sum 함수는 2개의 입력 값을 받아서 서로 더한 결과 값을 돌려준다.

따라서 sum = sum(3, 4)라고 하면 sum 함수 내에 parameter인 a, b에 각각 3과 4라는 argument를 받아 3과 4를 더한 결과인 7을 반환하게 된다. print()처럼 리턴 값이 없는 함수도 존재할 수 있고, 입력 값이 없는 함수, 입력 값과 리턴 값 모두가 없는 함수 또한 존재할 수 있다. 함수에서 중요한 점은 입력 값은 여러 개 일수 있지만, 결과값은 언제나 하나라는 점이다. 만약 결과 값을 반환하는데 2개 이상의 결과 값을 리턴한다면 결과 값들은 하나의 튜플에 모두 들어가 튜플값 하나로 돌려주게 된다. 만약 결과 값들을 각각 받고 싶다면 결과 값 수만큼의 변수에 대입해주면 된다. 그리고 함수는 return을 만나면 결과 값을 리턴하고 즉시 함수를 종료한다.

함수를 잘 사용하는 것이 중요한 이유는 프로그래밍 생산성을 높여준다는 것이다. 반복적인 작업을 수행할 때 직접 연산을 수행하도록 프로그램을 만들 수도 있지만, 간단한 연산이더라도 함수를 이용하면 수정해야 할 부분이 발생했을 때 함수 내의 연산 과정만 고치면 되기 때문에 보다 간단하게 문제를 해결할 수 있다. 그리고 함수를 적절히 사용하여 프로그램을 작성하면 결과를 내는 과정을 보다 간결하게 볼 수 있어서 오류를 찾아내기도 쉽고, 훨씬 체계적인 구조로 프로그래밍을 할 수 있다. 이러한 점은 코드가 길어지고 복잡해질수록 더욱 부각될 것이다. 즉, 프로그래밍에서 적절히 함수를 선언하고 잘 배치하는 것이 곧 프로그래머의 능력이라고 할 수 있다.