|  |
| --- |
|  |
| **Python\_3.X 기초** |
| **“점프 투 파이썬”을 읽고 나서…** |

목차

[1 개요 4](#_Toc520581212)

[1.1 프로젝트 주제 4](#_Toc520581213)

[1.2 프로젝트 추진 배경 및 목표 4](#_Toc520581214)

[2 왜 파이썬인가? 5](#_Toc520581215)

[3 간단한 입력과 출력 6](#_Toc520581216)

[3.1 입력과 출력 6](#_Toc520581217)

[4 프로그래밍의 기초, 자료형 7](#_Toc520581218)

[4.1 변수와 변수명 7](#_Toc520581219)

[4.2 자료형 8](#_Toc520581220)

[4.3 숫자형(Number) 8](#_Toc520581221)

[4.4 문자열 자료형 10](#_Toc520581222)

[4.4.1 문자열 연산하기 11](#_Toc520581223)

[4.4.2 문자열 인덱싱과 슬라이싱 12](#_Toc520581224)

[4.4.3 문자열 포매팅 14](#_Toc520581225)

[4.4.4 문자열 관련 함수 15](#_Toc520581226)

[4.5 불 자료형 17](#_Toc520581227)

[4.6 리스트 자료형 17](#_Toc520581228)

[4.6.1 리스트의 인덱싱과 슬라이싱 17](#_Toc520581229)

[4.6.2 리스트의 수정과 변경 18](#_Toc520581230)

[4.6.3 리스트 관련 함수 19](#_Toc520581231)

[4.7 튜플 자료형 20](#_Toc520581232)

[4.7.1 튜플의 인덱싱, 슬라이싱, +연산과 \*연산 20](#_Toc520581233)

[4.8 딕셔너리 자료형 21](#_Toc520581234)

[4.8.1 딕셔너리 관련 함수 22](#_Toc520581235)

[4.9 자료형 변환 23](#_Toc520581236)

[5 제어문 24](#_Toc520581237)

[5.1 연산자 24](#_Toc520581238)

[5.1.1 산술 연산자 24](#_Toc520581239)

[5.1.2 비교 연산자 25](#_Toc520581240)

[5.1.3 할당 연산자 25](#_Toc520581241)

[5.1.4 비트 연산자 25](#_Toc520581242)

[5.1.5 논리 연산자 26](#_Toc520581243)

[5.1.6 멤버 연산자 26](#_Toc520581244)

[5.1.7 식별 연산자 26](#_Toc520581245)

[5.2 if문 26](#_Toc520581246)

[5.3 for문 29](#_Toc520581247)

[5.3.1 range함수와 for문 30](#_Toc520581248)

[5.3.2 리스트에 for문 포함하기 31](#_Toc520581249)

[5.4 while문 32](#_Toc520581250)

[5.5 continue와 break문 34](#_Toc520581251)

[6 함수 34](#_Toc520581252)

[7 클래스 34](#_Toc520581253)

[8 모듈 34](#_Toc520581254)

[9 패키지 34](#_Toc520581255)

[10 참고자료 35](#_Toc520581256)

# 개요

## 프로젝트 주제

|  |
| --- |
| 개인 역량 강화 프로젝트 |

표 1- 프로젝트 주제

## 프로젝트 추진 배경 및 목표

|  |
| --- |
| 시큐어가드테크놀러지 입사 전 기초적인 python 문법을 익혀 역량을 강화해보자. |

표 1- 프로젝트 추진 배경 및 목표

# 왜 파이썬인가?

|  |
| --- |
| 문법을 쉽게 배우고 사용할 수 있고 가독성이 뛰어남. |
| 파이썬은 다른 언어로 만든 프로그램을 파이썬 프로그램에 포함시킬 수 있음. |
| 개발 속도가 빠름. |
| 교차 플랫폼 언어이기에 윈도우, 리눅스 등 다른 플랫폼에서도 동일하게 구동 가능. |
| 무료, 오픈소스이기에 자유롭게 파이썬을 이용 가능 |
| 파이썬으로 GUI(Graphical User Interface)를 개발 가능. |

표 2- 파이썬의 장점

|  |
| --- |
| 인터프리터 언어이기에 JAVA나 C++같은 컴파일 언어보다 느림. |
| 동시다발적 멀티스레드를 처리하거나 CPU에 집중된 많은 스레드를 처리하는 데 부적합. |
| 2.X 버전과3.X 버전이 호환되지 않음. |

표 2- 파이썬의 단점

# 간단한 입력과 출력

## 입력과 출력

**print() :** 입력한 자료를 출력한다.



그림 3- print 사용의 간단한 예

print(‘python’ ‘is’ ‘funny’)와 print(‘python’+‘is’+‘funny’)의 결과값은 같게 나온다.



그림 3- print에서 +와 따옴표 연산의 동일함을 보여주는 예

콤마(,)는 문자열의 띄어쓰기 역할을 한다. print(‘python’, ‘is’, ‘funny’)의 결과는 이렇다.



그림 3- print에서 콤마(,)의 역할

한 줄에 결과값을 출력할 수도 있다. for문을 이용해 예제를 만들어보았다.



그림 3- print에서 인수 end를 이용해 한 줄로 연결

(위 3-4 코드는 파이썬 2.7 버전의 경우 print(i, end=’ ‘)가 아닌 print i, 를 사용한다.)

**input() :** 사용자에게 입력을 받는다.



그림 3- input() 사용의 간단한 예

사용자에게 입력을 받을 때 안내문구 또는 질문을 출력하고 싶다면 input() 괄호 안에 내용을 입력해주면 내용을 출력한 후 입력을 받는다.



그림 3- 내용 출력 후 input()으로 5 입력

# 프로그래밍의 기초, 자료형

## 변수와 변수명

**변수 :** 객체를 가리키는 것이라고도 말할 수 있다. 객체란 자료형과 같은 것을 의미한다. 변수의 상징적인 이름을 데이터의 실제 위치로 치환해야 한다. 변수 값, 형, 위치는 일반적으로 고정된 채 유지되는 반면 위치에 저장되어 있는 데이터는 프로그램 실행 도중 변경될 수 있다

|  |
| --- |
| 영문자(대문자, 소문자), 숫자, 언더바(\_)를 사용할 수 있다. |
| 첫 자리에는 숫자를 사용할 수 없다. |
| 파이썬 키워드는 변수명으로 사용할 수 없다. |

표 4- 변수명의 규칙

파이썬에서 변수를 선언할 때 C, Java 등과 다르게 변수의 자료형과 함께 쓸 필요가 없다. 파이썬은 변수에 저장된 값을 스스로 판단하여 자료형을 알아낸다.

변수가 가리키는 메모리 주소를 id()함수로 알 수 있다. a 변수의 메모리 주소를 알아보자.



그림 4- 변수 a의 메모리 주소

## 자료형

**자료형 :** 프로그래밍을 할 때 쓰이는 숫자, 문자열 등 자료 형태로 사용하는 모든 것을 뜻한다. 프로그램의 기본이자 핵심 단위이다.

|  |  |
| --- | --- |
| 자료형 | 설명 |
| 정수형(Integer) | 양의 정수, 음의 정수, 0을 표현. Long은 데이터가 긴 경우. |
| 실수형(Floating-point) | 소수점이 포함된 숫자. |
| 복소수(Complex number) | 복소수, 실수와 허수부의 합으로 이루어진 수를 표현. |
| 문자열(String) | 문자, 단어 등으로 구성된 문자들의 집합을 의미. |
| 리스트(List) | 순서가 있는 값들의 나열. |
| 튜플(Tuple) | 리스트와 달리 요소의 값을 수정할 수 없음. |
| 딕셔너리(Dictionary) | key와 Value라는 것을 한 쌍으로 갖는 자료형. |
| 집합(Set) | 집합에 관련된 것들을 쉽게 처리하기 위해 만들어진 자료형 |
| 불(Boolean) | 참(True)과 거짓(False)을 나타내는 자료형. |

표 4- 자료형의 종류와 설명

## 숫자형(Number)

숫자형에는 정수형, 실수형이 존재한다.

|  |  |
| --- | --- |
| 항목 | 사용 예시 |
| 정수 | 123, -456, 0 |
| 실수 | 1.23, -45.67, 3.4e10 |

표 4- 자료형의 종류와 설명

표 4-3에서 실수형의 예시 중 3.4e10는 무엇일까?

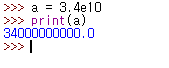


그림 4- e(지수 표현 방식)의 사용 예

3.4e10은 3.4 \* 10^10을 의미한다. 즉, e뒤에 수는 10의 지수를 의미한다.

만약 정수/정수의 값이 실수의 형태를 가지면 어떻게 출력될까?



그림 4- 3/4의 결과값

파이썬 3.X버전에서는 값이 0.75로 나오지만, 2.X 버전에서는 정수형으로 나눌 경우 정수로만 결과값을 리턴하기 때문에 0이 출력된다.

위에서 /는 실수를 리턴했지만 // 연산자를 사용하면 소수점 아랫자리를 버리고 정수 부분만 리턴한다.

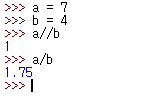


그림 4- // 연산과 /연산의 차이

여기서 주의할 점은 음수를 연산할 시 값이 예상과 다를 수 있다는 점이다. a를 -7로 바꾸어 연산해보자.

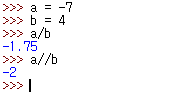


그림 4- 음수일 때 // 연산

//연산자는 나눗셈의 결과에서 무조건 소수점을 버리는 것이 아니라 나눗셈의 결과값보다 작은 정수 중, 가장 큰 정수를 리턴하기 때문이다. -1.75 > -2 이므로 결과값보다 작은 수 중에서 가장 큰 정수이다.

제곱 연산자도 존재한다. a\*\*b는 a^b를 뜻한다.

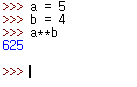


그림 4- \*\*연산자의 사용

a\*\*b의 결과값은 a의 네 제곱인 625가 리턴된다.

파이썬은 복소수를 지원한다. 사용법은 j를 쓰면 된다.

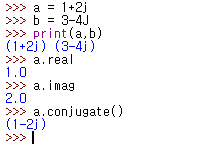


그림 4- a를 여러 방법으로 출력

print(a, b)를 실행하면 a와 b의 켤레복소수를 출력한다. a.conjugate()도 역시 켤레복소수를 리턴한다. a.real은 복소수의 실수 부분을 리턴한다. a.imag은 복소수의 허수부분을 리턴한다.

## 문자열 자료형

문자열을 생성할 때는 4가지 방법을 사용할 수 있다.

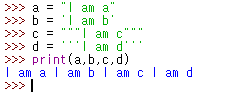


그림 4- 문자열을 생성하는 예

큰따옴표로 감싼 문자열 안에 있는 작은따옴표는 문자열을 나타내기 위한 기호로 인식되지 않는다. 또한 작은따옴표로 감싼 문자열 안에 있는 큰 따옴표도 문자열을 나타내기 위한 기호로 인식되지 않는다.

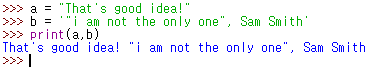


그림 4- 쌍따옴표(“ ”)와 따옴표(‘ ‘)의 사용 예

백슬래시(\)를 이용해 ‘와 “를 문자열에 포함시킬 수 있다.

|  |  |
| --- | --- |
| 코드 | 설명 |
| \n | 개행(줄바꿈) |
| \t | 수평 탭 |
| \\ | 문자 “\” |
| \’ | 작은따옴표 그대로를 출력 |
| \” | 큰따옴표 그대로를 출력 |
| \r | 캐리지 리턴 |
| \f | 폼 피드 |
| \a | 벨 소리 |
| \b | 백 스페이스 |
| \000 | 널 문자 |

표 4- 이스케이프 코드의 종류

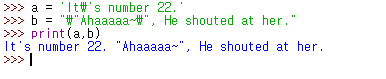


그림 4- \의 사용 예

### 문자열 연산하기

파이썬에서는 문자열을 + 기호로 더하여 연결할 수 있다.

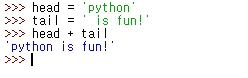


그림 4- +로 문자열 연결하기

또한 \*로 문자열을 곱하여 출력할 수도 있다.

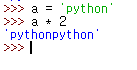


그림 4- \*로 문자열 반복하기

\* 연산을 사용하면 문자열을 그 수만큼 반복한다는 의미이다. 이 기능만 봐서는 사용할 상황이 떠오르지 않을 것이다. 아래와 같이 콘솔을 간단하게 디자인 할 수 있다.

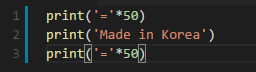


그림 4- 문자열에서 \*연산의 사용 예

이 코드의 실행 결과는 이렇다.

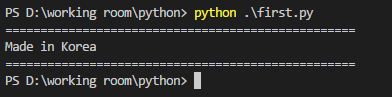


그림 4- [그림 4- 13]의 결과

=를 50개만 출력했을 뿐인데 훨씬 보기 좋아진 기분이다.

### 문자열 인덱싱과 슬라이싱

**인덱싱(Indexing) :** 어떤 것을 가리킨다는 의미이다.

**슬라이싱(Slicing) :** 어떤 것을 잘라낸다는 의미이다.

문자열 인덱싱의 예를 살펴보자.

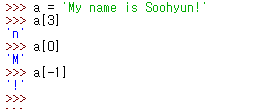


그림 4- 인덱싱의 표현 방식

a[3]은 문자열 a의 맨 앞(a[0])에서부터 3칸 떨어진 n을 리턴한다. a[0]은 맨 앞의 M을 리턴한다. 만약 위치가 a[-1]처럼 음수일 경우에는 뒤에서부터 차례대로 리턴한다는 의미이다. a[-1]은 맨 뒤에서 첫번째인 !를 리턴한다.

위에서는 문자열의 요소 중 하나만을 뽑아내었다. 그렇다면 ‘name’이나 ‘Soohyun’과 같이 단어를 뽑아내는 방법이 바로 슬라이싱이다.

‘name’을 인덱싱과 슬라이싱으로 추출해보자.

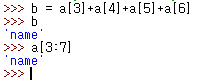


그림 4- 인덱싱과 슬라이싱의 예

인덱싱은 a[3]부터 a[6]까지 더하였다. 그렇다면 슬라이싱은 왜 a[3:7]로 선언했을까? 슬라이싱은 a[x:y]라고 한다면 x<=n<y인 수식으로 슬라이싱을 한다. 3<=n<7을 만족하는 a[]는 a[3] ~ a[6]이다.

슬라이싱 기법을 사용하기에 좋은 예시를 들어보자.

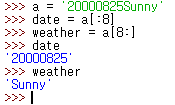


그림 4- 슬라이싱의 좋은 사용 예

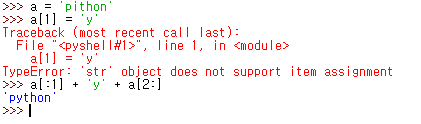


그림 4- 문자열 수정 후에 출력하기

문자열은 요소의 값을 바꿀 수 없다. 그렇기에 a[0]과 a[2] 사이에 ‘y’라는 문자를 추가한 후 이어 붙여서 만든다.

### 문자열 포매팅

**문자열 포매팅 :** 문자열 내에 어떠한 값을 삽입하는 것을 말한다.

|  |  |
| --- | --- |
| 코드 | 설명 |
| %s | 문자열(String) |
| %c | 문자(Character) |
| %d | 정수(Integer) |
| %f | 부동소수(floating-point) |
| %o | 8진수 |
| %x | 16진수 |
| %% | 리터럴 % (문자 % 그대로) |

표 4- 문자열 포맷 코드

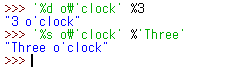


그림 4- 문자열 포매팅의 예

정수나 실수, 문자열, 변수 등을 포함시킬 수 있다. 아래처럼 한번에 여러 개의 값을 삽입할 수 있다. 또한 연산도 할 수 있다.



그림 4- 한 문자열에 여러 개의 포매팅

format 함수를 이용해 좀 더 발전된 스타일로 문자열 포맷을 지정할 수 있다.

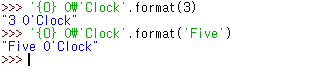


그림 4- format()함수의 사용 예

{0} 다음은 {1}, {2}… 순서로 format()함수를 이용하여 입력도 가능하고 이름을 지어서 대입하는 방법으로도 입력할 수 있다.

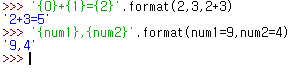


그림 4- format()함수의 다른 입력 방법

파이썬 3.6 버전 이상에서는 f 문자열 포매팅을 사용할 수 있다. 사용법은 문자열 앞에 f 접두사를 붙인다. 이 기능을 이용해 왼쪽이나 오른쪽, 가운데로 정렬하는 것 또한 가능하다. 정렬하여 생기는 공백을 채울 수도 있다.

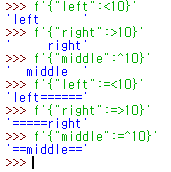


그림 4- 좌, 우, 가운데 정렬 및 공백 채우기

### 문자열 관련 함수

|  |  |  |  |
| --- | --- | --- | --- |
| 함수명 | 기능 | 함수명 | 기능 |
| count() | 문자의 개수 반환 | lstrip() | 왼쪽 공백 지우기 |
| find() | 위치 알려줌 | rstrip() | 오른쪽 공백 지우기 |
| index() | 위치 알려줌 | strip() | 양쪽 공백 지우기 |
| join() | 문자열 삽입 | replace | 문자열 바꾸기 |
| upper() | 소문자 => 대문자 | split | 문자열 나누기 |
| lower() | 대문자 => 소문자 |  |  |

표 4- 문자열과 관련된 함수 종류

표의 왼쪽 위의 함수부터 실습해보자.



그림 4- count()함수 사용의 예

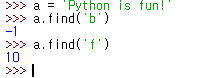


그림 4- find()함수의 사용 예. -1은 찾는 값이 존재하지 않음을 의미

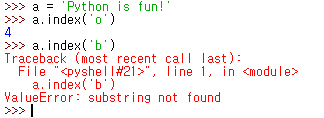


그림 4- index()함수의 사용 예. 오류 메시지는 찾는 값이 존재하지 않음을 의미



그림 4- join()함수의 사용 예



그림 4- upper()함수의 사용 예



그림 4- lower()함수의 사용 예

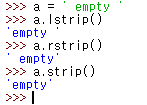


그림 4- lstrip(), rstrip(), strip()함수의 사용 예



그림 4- replace()함수의 사용 예

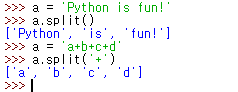


그림 4- split()함수의 사용 예

## 불 자료형

불(bool) 자료형은 참(True)과 거짓(False) 두 가지 값만을 가질 수 있다.

|  |  |  |  |
| --- | --- | --- | --- |
| 값(유형) | 참 or 거짓 | 값 | 참 or 거짓 |
| “python” | 참 | {} | 거짓 |
| “” | 거짓 | 1 | 참 |
| [1,2,3] | 참 | 0 | 거짓 |
| [] | 거짓 | None | 거짓 |
| () | 거짓 |  |  |

표 4- 값들의 참과 거짓

## 리스트 자료형

|  |
| --- |
| a = [ ] |
| b = [1, 2, 3,] |
| c = [‘python’, ‘is’, ‘fun’] |
| d = [1, 2, ‘python’, ‘is’] |
| e = [1, 2, [‘python’, ‘is’]] |

표 4- 리스트 자료형의 사용 예

### 리스트의 인덱싱과 슬라이싱

리스트에서도 문자열처럼 인덱싱과 슬라이싱 기능을 지원한다.

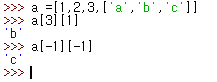


그림 4- 리스트 인덱싱의 사용 예



그림 4- 리스트 슬라이싱의 사용 예

### 리스트의 수정과 변경

리스트에서는 문자열과 다르게 요소를 수정할 수 있다.

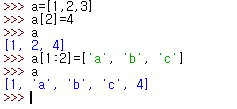


그림 4- 리스트의 값 수정

여기서 주의해야 할 점은 연속된 범위의 값을 수정할 경우 위(a[1:2]=[‘a’,’b’,’c’])와 같이 슬라이싱으로 수정해주어야 한다. 만약 a[1]에 대입한다면 어떻게 될까?

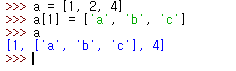


그림 4- 이중 리스트로 수정

a[1] 주소에 연속된 값이 리스트 형태로 입력되어 이중 리스트 형태를 가졌다. a[1]과 a[1:2]의 차이점을 알고 사용해야 한다.

리스트의 요소의 값을 수정만 할 수 있는 것이 아니라 삭제도 가능하다.

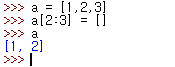


그림 4- 리스트의 요소 삭제



그림 4- del함수를 이용한 리스트의 요소 삭제

### 리스트 관련 함수

|  |  |  |  |
| --- | --- | --- | --- |
| 함수명 | 기능 | 함수명 | 기능 |
| append | 리스트에 요소 추가 | remove | 리스트 요소 제거 |
| sort | 리스트 정렬 | pop | 리스트 요소 꺼내기 |
| reverse | 리스트 뒤집기 | count | 요소 개수 세기 |
| index | 위치 반환 | extend | 리스트 확장 |
| insert | 리스트에 요소 삽입 |  |  |

표 4- 리스트와 관련된 함수 종류

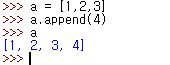


그림 4- append()함수의 사용 예

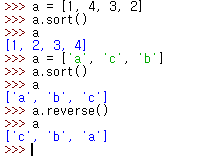


그림 4- sort()함수와 reverse()함수의 사용 예

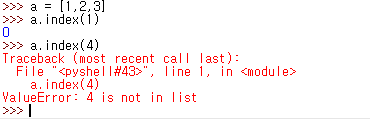


그림 4- index()함수의 사용 예, 값이 존재하지 않을 경우 오류 출력

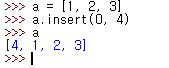


그림 4- insert()함수의 사용 예

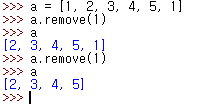


그림 4- remove()함수의 사용 예

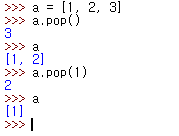


그림 4- pop()함수의 사용 예



그림 4- count()함수의 사용 예

## 튜플 자료형

튜플은 리스트와 비슷하지만 리스트와는 다르게 요소의 값을 수정, 삭제할 수 없다.

튜플은 다음과 같은 형태이다.

|  |
| --- |
| t1 = () |
| t2 = (1, ) |
| t3 = (1, 2, 3) |
| t4 = 1, 2, 3 |
| t5 = (‘a’, ‘b’, (‘ab’, ‘cd’)) |

표 4- 튜플 자료형의 사용

### 튜플의 인덱싱, 슬라이싱, +연산과 \*연산

튜플은 문자열, 리스트와 마찬가지로 인덱싱과 슬라이싱이 가능하다. 또한 튜플을 더하거나 곱할 수 있다.

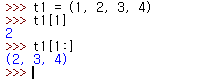


그림 4- 튜플의 인덱싱과 슬라이싱

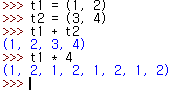


그림 4- 튜플의 +연산과 \*연산

## 딕셔너리 자료형

딕셔너리는 key와 Value라는 것을 한 쌍으로 갖는 자료형이다. key:value 는 대응 관계를 나타내는데, 이를 연관 배열(Associative array) 또는 해시(Hash)라고 한다.



|  |  |
| --- | --- |
| Key | Value |
| name | soo |
| years | 19 |
| birthday | 000825 |

표 4- dic의 Key와 Value 분류

딕셔너리는 key로 그에 대응하는 value를 얻을 수 있다.

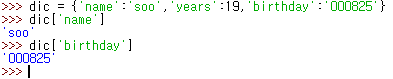


그림 4- key에 대응하는 value를 얻기

딕셔너리에 쌍을 추가하거나 삭제하는 것이 가능하다.

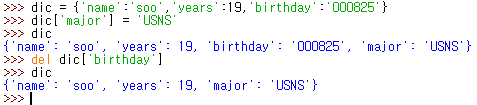


그림 4- 딕셔너리의 요소 추가, 삭제

### 딕셔너리 관련 함수

|  |  |  |  |
| --- | --- | --- | --- |
| 함수명 | 기능 | 함수명 | 기능 |
| keys | key 리스트 생성 | clear | 쌍 모두 지우기 |
| values | value 리스트 생성 | get | key로 value 얻기 |
| items | key, value 쌍 얻기 | in | 특정key의 존재 여부 |

표 4- 딕셔너리와 관련된 함수 종류

딕셔너리 a는 {‘name’ : ‘pey’, ‘phone’ : ‘0119993323’, ‘birth’ : ‘1118’} 이다.

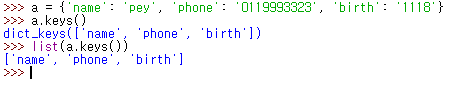


그림 4- keys()함수의 사용 예

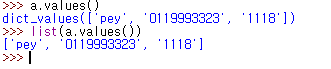


그림 4- values()함수의 사용 예



그림 4- items()함수의 사용 예



그림 4- clear()함수의 사용 예

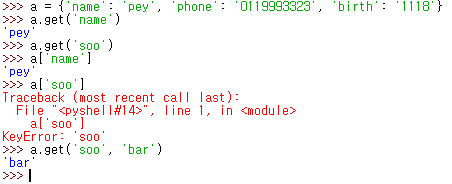


그림 4- get()함수의 사용 예

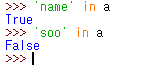


그림 4- in 함수의 사용 예

## 자료형 변환

위에서 다룬 내용이 자료형이었다. 만약 내가 사용한 숫자나 변수의 자료형이 무엇인지 궁금하다면 type()함수를 사용하여 자료형을 알 수 있다.

type()함수의 사용법은 이렇다.



그림 4- type()함수를 사용해 자료형을 출력

자료형이 어떤 것인지 파악하는 것은 정말 중요하다. input()함수로 예를 들어보자.

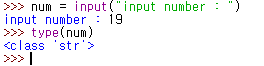


그림 4- input()함수의 자료형

분명히 정수인 19를 입력 받았음에도 불구하고 자료형은 문자열이다. 만약 num을 정수형과 더하면 어떻게 될까?

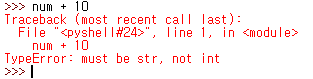


그림 4- input()함수와의 연산 오류

문자열과 정수형은 더할 수 없으므로 오류를 출력한다. 이러한 상황에 필요한 것은 형변환이다. 다음은 형변환으로 위 문제를 해결하는 두 가지의 방법이다.



그림 4- num을 형변환하여 오류 수정

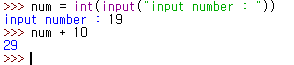


그림 4- input 자체를 int형으로 변환

# 제어문

## 연산자

### 산술 연산자

a = 10, b = 20, c = 3 일 때,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 코드 | 기능 | 예시 | 코드 | 기능 | 예시 |
| + | 더하기 | a + b = 20 | **%** | 나머지 | b % a = 0 |
| - | 빼기 | a – b = -10 | **\*\*** | 제곱 | a \*\* c =1000 |
| \* | 곱하기 | a \* b =200 | **//** | 몫 | a // c = 3 |
| / | 나누기 | b / a = 2.0 |  |  |  |

표 5- 산술 연산자의 종류

### 비교 연산자

a = 10, b =20 일 때,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 코드 | 기능 | 예시 | 코드 | 기능 | 예시 |
| == | 값이 동일 | (a==b) False | **<** | 좌가 우보다 작음 | (a<b) True |
| != | 값이 다름 | (a!=b) True | **>=** | // 크거나 같음 | (a>=b) False |
| > | 좌가 우보다 큼 | (a>b) False | **<=** | // 작거나 같음 | (a<=b) True |

표 5- 비교 연산자의 종류

### 할당 연산자

a = 10, b = 20일 때,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 코드 | 기능 | 예시 | 코드 | 기능 | 예시 |
| = | 좌 변수에 우의 값을 할당 | c = a + b c = a + b | **/=** | 좌 변수의 우의 값을 나누어 할당 | c /= a  c = c / a |
| += | 좌 변수에 우의 값을 더하여 할당 | c += a  c = c + a | **%=** | 좌 변수의 우의 값을 나눈 나머지를 할당 | c %= a  c = c % a |
| -= | 좌 변수에 우의 값을 빼어 할당 | c -= a  c = c - a | **\*\*=** | 좌 변수에 우의 값을 제곱하여 할당 | c \*\*= a → c = c \*\* a |
| \*= | 좌 변수에 우의 값을 곱하여 할당 | c \*= a  c = c \* a | **//=** | 좌 변수에 우의 값을 나눈 몫을 할당 | c //= a → c = c // a |

표 5- 할당 연산자의 종류

### 비트 연산자

a = 59, b = 26 일 때, 이진수로 변환할 시 a = 0011 1011, b = 0001 1010

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 코드 | 기능 | 예시 | 코드 | 기능 | 예시 |
| & | AND 연산. 둘 다 참일 때 1 | (a&b) = 26  0001 1010 | **~** | 보수 연산. | (~a) = -59  1100 0100 |
| | | OR 연산. 둘 중 하나 이상이 참일 때 1 | (a|b) = 59  0011 1011 | **<<** | 좌 시프트 연산자 | (a<<2) = 236  1110 1100 |
| ^ | XOR 연산. 둘 중 하나만 참일 때 1 | (a^b) = 33  0010 0001 | **>>** | 우 시프트 연산자 | (a>>2) = 14  0000 1110 |

표 5- 비트 연산자의 종류

### 논리 연산자

a = True, b = False일 때,

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 코드 | 기능 | 예시 | 코드 | 기능 | 예시 |
| and | 둘 다 참일 때 참 | a and b = False | **not** | 논리 상태를 반전 | not(a and b) = True |
| or | 둘 중 하나 이상이 참일 때 참 | a or b = True |  |  |  |

표 5- 논리 연산자의 종류

### 멤버 연산자

a = 10, b = 10, list = [1, 2, 3, 4, 5]일 때,

|  |  |  |
| --- | --- | --- |
| 코드 | 기능 | 예시 |
| in | list 내에 포함되어 있으면 참 | (a in list) = False |
| not in | list 내에 포함되어 있지 않으면 참 | (b not in list) = True |

표 5- 멤버 연산자의 종류

### 식별 연산자

a = 20, b = 20일 때,

|  |  |  |
| --- | --- | --- |
| 코드 | 기능 | 예시 |
| is | 개체의 메모리 위치나 값이 같다면 참 | (a is b) = True |
| is not | 게체의 메모리 위치나 값이 같지 않다면 참 | (a is not b) = False |

표 5- 식별 연산자의 종류

## if문

조건문 중 하나인 if문은 조건이 참일 경우 if문 내의 코드 블록을 실행한다. 그렇다면 if에서 참일 경우는 어떤 것이 있을까?

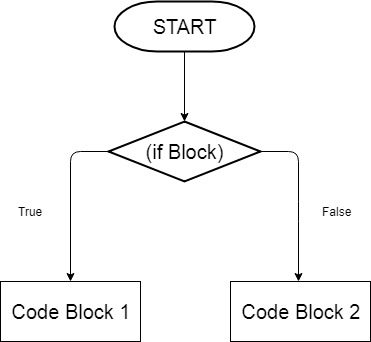
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| 자료형 | 참 | 거짓 | 자료형 | 참 | 거짓 |
| 숫자 | 0이 아닌 숫자 | 0 | **튜플** | (1,2,3) | () |
| 문자열 | "abc" | "" | **딕셔너리** | {"a":"b"} | {} |
| 리스트 | [1,2,3] | [] | **튜플** | (1,2,3) | () |

표 5- 자료형 별 참과 거짓

일반적으로 참은 자료형에 값이 들어있는 경우가 대부분이고 거짓은 반대로 자료형에 값이 없는 경우이다.

if문을 사용하기 위하여 if문의 순서도와 구조를 알아보자.

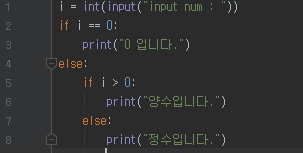
|  |
| --- |
| **if** 조건문 **:**  수행할 문장1  수행할 문장2  …  **else :**  수행할 문장1  수행할 문장 2  … |

코드 5- if문의 기본 구조

순서도 5- if문의 순서도

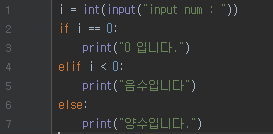
if 조건문: 이 참이라면 그 안에 있는 코드 블록 1을 실행하고 거짓이라면 코드 블록 2를 실행한다. 파이썬은 다른 언어와 다르게 올바른 들여쓰기를 하지 않으면 오류가 발생하거나 코드의 흐름이 바뀔 수도 있기 때문에 꼭 위 코드처럼 들여쓰기를 하는 습관을 가져야 한다.

if는 참일 때, else는 if 조건에 맞지 않을 경우 실행하는 것이라면, 다중 if문을 사용하지 않고 여러 조건식을 사용할 수도 있다. 바로 elif를 사용해서 가능하다.



코드 5- 다중 if문을 사용한 코드

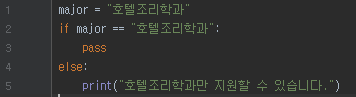
즉, 이렇게 다중 if문을 사용하였던 코드를 elif를 사용하여 같은 기능을 수행하도록 만들어보자.



코드 5- elif문을 사용한 코드

다중 if문을 사용할 때보다 가독성도 뛰어나며 비교적 성능의 저하도 해결할 수 있다.

이번에는 if-else문을 사용해 참일 경우 아무런 일도 하지 않게 설정하는 법이다.



코드 5- pass함수를 사용한 코드

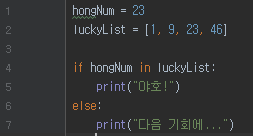
pass함수는 코드를 단순히 진행만 하게 만드는 역할이다. 만약 major가 “호텔조리학과”라면 아무 코드도 실행하지 않는 것이다. pass함수는 continue와 비슷하지만 엄연히 다른 결과를 가진다. pass함수는 단순히 실행할 코드가 없다는 것을 의미하지만, continue는 다음 순번의 루프로 강제로 이동하는 것이다. continue는 차후에 자세히 언급할 것이다.

예제로 if문의 사용을 익혀보자.

1. 홍길동씨의 행운권 번호는 23번 이라고 한다. 다음은 행운권 당첨번호 리스트이다.

luckyList = [1, 9, 23, 46]

홍길동씨가 당첨되었다면 “야호”라는 문자열을 출력하는 프로그램을 작성하시오.



코드 5- if문 예제

in 함수는 리스트, 튜플, 문자열 자료형에 찾는 값이 존재하면 참을 반환하는 것을 이용하여 문제를 해결하였다.

## for문

반복문 중의 하나로, 문장 구조를 쉽게 이해할 수 있다는 장점이 있다.

|  |
| --- |
| **for** 변수 **in** (리스트 or 튜플 or 문자열)**:**  수행할 문장1  수행할 문장 2  … |

코드 5- for문의 기본 구조

리스트나 튜플, 문자열의 첫 번째 요소부터 마지막 요소까지 차례로 변수에 대입되어 코드 블록을 수행한다.

for문의 사용 예를 알아보자.

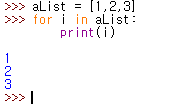


그림 5- for문의 사용 예

i에 aList의 요소값을 대입한 후 주어진 문장인 print(i)를 실행한다. 이것을 마지막 요소까지 반복한다.

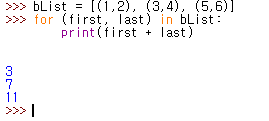
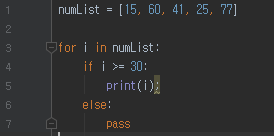


그림 5- for문의 사용 예

bList에 요소값으로 튜플을 대입했다. 튜플의 요소가 둘이므로 대입할 변수를 first와 last 둘로 선언하였다. 그 후 차례대로 대입한 후 문장을 실행하면 리스트 내의 튜플의 합을 출력한다.

리스트와 for문을 이용하여 정수로 이루어진 리스트의 요소 중 30 이상인 수를 출력해보자.



코드 5- 리스트 내의 30 이상인 수 출력을 위한 코드

이런 방식으로 for문을 사용할 수 있다. 실행 결과는 60과 41, 77을 순서대로 출력한다.

### range함수와 for문

range는 숫자 리스트를 자동으로 만들어주는 함수이다. range는 for문과 함께 사용하는 경우가 잦다.

|  |
| --- |
| 변수명 = **range**((시작할 수), (끝낼 수(미만))) |

코드 5- range()함수의 사용 구조

만약 range(1,4)라면, 1부터4미만인 리스트를 만든다. 또 range(7)과 같이 인자를 하나만 입력했을 경우 0부터 7미만인 리스트를 만든다.

for문과 range를 함께 사용해 1부터 10까지의 정수의 합을 구해보자.

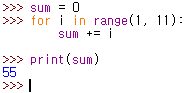
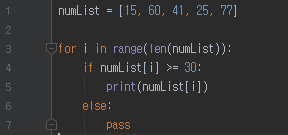


그림 5- for문과 range를 혼용한 1부터 10까지의 합

변수 i에 1부터 10까지의 수를 차례대로 대입한 후 sum에 더해진다.

range함수를 사용하여 리스트 요소 중 30 이상의 요소를 출력해보자.



코드 5- range를 사용한 리스트 내의 30 이상인 수 출력을 위한 코드

len()함수는 리스트의 길이를 반환하는 기능을 가지고 있다. 변수 i에0부터 리스트의 길이 미만까지 i에 대입한다. 만약 numList의 i번째(0 포함) 요소의 값이 60이 넘는다면 그 값을 출력하고 그렇지 않다면 다음 코드를 진행한다. 결과는 60과 41, 77을 출력한다.

### 리스트에 for문 포함하기

리스트 내에 for문을 포함할 수도 있다.

|  |
| --- |
| 변수명1 = [표현식 **for** 변수명2 **in** 반복가능객체 **if** 조건1  **for** 변수명? **in** 반복가능객체 **if** 조건2  … |

코드 5- 리스트 내포의 구조

이 내용을 설명하기 전에 리스트 내에 포함시키지 않고 for문을 사용한 예를 보자.

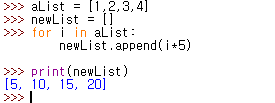


그림 5- for문을 이용하여 aList의 요소를 5 곱한 값을 newList에 저장

위 내용은 aList의 요소를 5로 곱하여 그 값을 newList에 저장한다. 이것을 리스트 내포를 이용해 만들어보자.



그림 5- 리스트 내포를 사용한 예

여기에 if문을 추가하여 짝수에만 5를 곱하여 리스트에 삽입해보자.



그림 5- 조건이 있는 리스트 내포를 사용한 예

출력 결과를 보니 aList = [1,2,3,4]에서 짝수인 2와 4만 5를 곱하여 newList에 저장하는데 성공하였다.

## while문

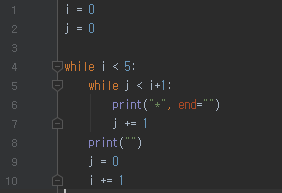
반복문 중 하나인 while문은 조건문이 참일 경우 문장을 반복한다. 즉, 조건이 거짓이 될 때 반복 수행을 중지한다.

|  |
| --- |
| **while** (조건문):  수행할 문장1  수행할 문장2  수행할 문장3  … |

코드 5- while문의 기본 구조

while문은 증감식을 따로 써주는 구간이 없기 때문에 수행할 문장에 증감식을 써 주지 않으면 조건문이 참일 때 끝나지 않을 수도 있다. (무한 루프)

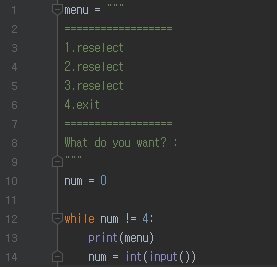
while문으로 간단한 예제를 실습해보자. 다음은 별의 개수를 한 줄 당 하나씩 늘려가며 마지막 5번째 줄에는 5개를 출력하는 예제이다.



코드 5- 별 찍기 예제

j = 0으로 초기화한 이유는 while j < i+1에서 거짓이 될 경우 j의 값이 증가한 채로 값이 유지되기 때문이다. i는 줄을 뜻하고 j는 별의 개수를 뜻한다.

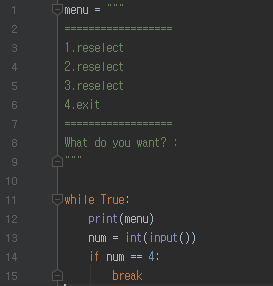
다음 예제는 4를 입력할 시 종료되는 프로그램이다.



코드 5- 4를 입력하면 종료하는 프로그램 1

num != 4 라는 조건식이 거짓이면 프로그램을 종료하는데, 이는 num에 4를 입력해야 종료한다는 의미이다. 만약 4가 아닌 다른 수를 입력한다면 4를 입력할 때까지 무한히 실행한다.

위 코드를 다른 방식으로도 작성할 수 있다.



코드 5- 4를 입력하면 종료하는 프로그램 2

위 두 개의 코드의 기능은 같다. 하지만 코드의 스타일은 개인차가 있으므로 이런 예도 들어보았다.

## continue와 break문

# 함수

# 클래스

# 모듈

# 패키지

# 참고자료

김왼손의 미운코딩새끼: 4시간만에 끝내는 파이썬 기초 (<https://www.inflearn.com/course/%ED%8C%8C%EC%9D%B4%EC%8D%AC-%EA%B8%B0%EC%B4%88-%EA%B0%95%EC%A2%8C/>)

점프 투 파이썬 (<https://wikidocs.net/book/1>)

점프 투 파이썬 (이지스퍼블리싱 출판, 박응용 지음)

Python 3.4 공부 좀 해볼까? (<https://wikidocs.net/book/122>)