



PYTHON (1)



🔍 "Life is too short, You need Python!"

운영진 김지안



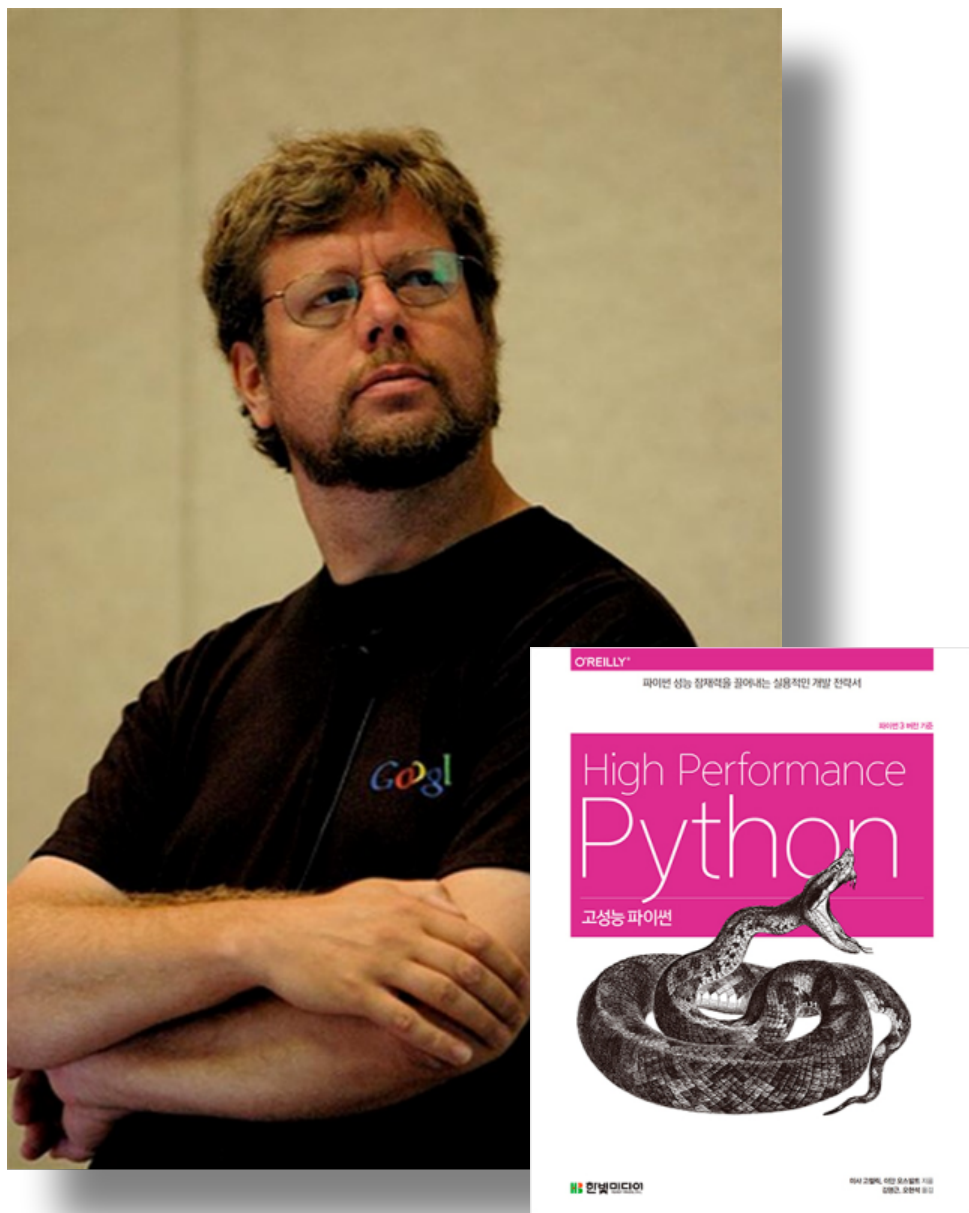


Python 소개



파이썬 소개 및 특징 설명

01 파이썬이란?



- 암스테르담의 귀도 반 로섬(Guido Van Rossum)이 개발하여 1991년에 발표된 프로그래밍 언어
- 파이썬이란 이름은 귀도 반 로섬이 당시 좋아했던 코디미 쇼인 '몬티 파이썬의 날아다니는 서커스'에서 따왔다.
- 파이썬의 사전적 의미는 고대 신화 속 아폴로 신이 델파이에서 물리친 큰 뱀의 이름으로, 대부분의 파이썬 책 표지가 뱀인 이유이다.
- 프로그래밍 언어 중에서도 오래전에 만들어진 '포트란'은 1954년에 탄생했고, 지금도 많이 사용하는 C언어는 1972년에 공개되었다. 파이썬은 비교적 최근에 개발된 언어이다.

02 파이썬특징

- 인간친화적인 프로그래밍 언어이다.
- 문법이 쉬워 빠르게 읽을 수 있다.
 - *유명한 프로그래머 에릭 레이먼(Eric Raymond)는 파이썬을 공부한 지 단 하루 만에 자신이 원하는 프로그램을 작성할 수 있었다고 한다.
- 오픈소스인 파이썬은 무료이지만 만들고자 하는 대부분의 프로그램을 만들 수 있어 강력하다.
- 간결하다.
 - *복잡한 프로그래밍 언어 펄(Perl)이 한가지를 구현하는 100가지 방법을 지원한다면, 파이썬은 가장 좋은 한가지 방법만 사용하는 것을 좋아한다.
- 개발 속도가 빠르다.

"Life is too short, You need Python!"

03 파이썬으로 할 수 있는 일 vs 할 수 없는 일

- 시스템 유틸리티 제작
(컴퓨터 사용에 도움을 주는 여러 소프트웨어)
- GUI 프로그래밍
(화면에 또 다른 윈도우 창을 만들고 그 창에 프로그램을 동작시킬 수 있는 메뉴나 버튼, 그림 등을 추가하는 것)
- C/C++과 결합
- 웹 프로그래밍
- 수치 연산 프로그래밍(Numpy 모듈)
- 데이터베이스 프로그래밍
- 데이터 분석, 사물인터넷

- 시스템과 밀접한 프로그래밍 영역
- 모바일 프로그래밍(가능은 하지만 python 만으로 만들기엔 아직 역부족이라고 한다.)



자료형

숫자형, 문자열, 리스트, 튜플, 딕셔너리



01 숫자형

- 정수형(Integer)

```
>>> a = 123
```

```
>>> a = -178
```

```
>>> a = 0
```

- 실수형(Floating-point)

```
>>> a = 1.2
```

```
>>> a = -3.45
```

- 8진수(Octal)

```
>>> a = 0o177
```

- 16진수(Hexadecimal)

```
>>> a = 0x8ff
```

```
>>> b = 0xABC
```

- 사칙연산

```
>>> a = 3
```

```
>>> b = 4
```

```
>>> a + b
```

```
7
```

```
>>> a - b
```

```
-1
```

```
>>> a * b
```

```
12
```

```
>>> a / b
```

```
0.75
```

01 숫자형

- x의 y제곱을 나타내는 ** 연산자
- 나눗셈 후 나머지를 반환하는 % 연산자
- 나눗셈 후 몫을 반환하는 // 연산자

```
>>> a = 3
>>> b = 4
>>> a ** b
81
```

```
>>> 7 % 3
1
>>> 3 % 7
3
```

	2	
3	7	
	6	
	1	→ 나머지

```
>>> 7 / 4
1.75
>>> 7 // 4
1
```

	1	→ 몫
4	7	
	4	
	3	

02 문자열(String)

- 문자열을 만드는 방법

- (1) 큰 따옴표(")로 양쪽 둘러싸기 `"Hello World"`
- (2) 작은 따옴표(')로 양쪽 둘러싸기 `'Python is fun'`
- (3) 큰 따옴표 연속 3개로 양쪽 둘러싸기 `"""Life is too short, You need python"""`
- (4) 작은 따옴표 연속 3개로 양쪽 둘러싸기 `'''Life is too short, You need python'''`

- 문자열 안에 작은 따옴표나 큰 따옴표를 포함시키고 싶을 때

- (1) 작은 따옴표를 포함시키고 싶다면 큰 따옴표로 둘러싸기 `"Python's favorite food is perl"`
- (2) 큰 따옴표를 포함시키고 싶다면 작은 따옴표로 둘러싸기 `'Python is very easy.' he says.'`
- (3) 백슬래시(\) 사용하기 `'Python\'s favorite food is perl' "\"Python is very easy.\" he says.'`
- (4) 여러 줄인 문자열을 변수에 대입하고 싶을 때

```
>>>multiline=""" #작은 따옴표도 가능
... Life is too short
... You need python
... """
```

02 문자열(String)

- 이스케이프 코드 : 주로 출력물을 보기 좋게 정렬하는 용도로 사용하는, 미리 정의해둔 문자조합(ex.\n)

코드	설명
<code>\n</code>	문자열 안에서 줄을 바꿀 때 사용
<code>\t</code>	문자열 사이에 탭 간격을 줄 때 사용
<code>\\</code>	문자 <code>\</code> 를 그대로 표현할 때 사용
<code>\'</code>	작은따옴표(')를 그대로 표현할 때 사용
<code>\"</code>	큰따옴표(")를 그대로 표현할 때 사용

<code>\r</code>	캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)
<code>\f</code>	폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)
<code>\a</code>	벨 소리(출력할 때 PC 스피커에서 '뽵' 소리가 난다)
<code>\b</code>	백 스페이스
<code>\000</code>	널 문자

※ \n, \t, \\, \', \"를 주로 사용하고 나머지는 거의 사용하지 않는다.

02 문자열(String)

- 문자열 연산하기

```
>>> head = "Python"
>>> tail = " is fun!"
>>> head + tail
'Python is fun!'
```

```
>>> a = "python"
>>> a * 2
'pythonpython'
```

- 문자열 길이 구하기

```
>>> a = "Life is too short"
>>> len(a)
17
```

- 문자열 인덱싱

```
>>> a = "Life is too short, You need Python"
>>> a[0]
'L'
>>> a[12]
's'
>>> a[-1]
'n'
```

```
>>> a = "Life is too short, You need Python"
>>> b = a[0] + a[1] + a[2] + a[3]
>>> b
'Life'
```

02 문자열(String)

- 문자열 슬라이싱

```
>>> a = "Life is too short, You need Python"
>>> a[0:4] 0 <= a < 4
'Life'
```

```
>>> a[0:2]
'Li'
>>> a[5:7]
'is'
>>> a[12:17]
'short'
```

```
>>> a[:17]
'Life is too short'
```

```
>>> a[19:-7]
'You need'
```

```
>>> a[:]
'Life is too short, You need Python'
```

- 문자열 포매팅(Formatting)

```
>>> number = 3
>>> "I eat %d apples." % number
'I eat 3 apples.'
```

```
>>> number = 10
>>> day = "three"
>>> "I ate %d apples. so I was sick for %s days." % (number, day)
'I ate 10 apples. so I was sick for three days.'
```

코드	설명	%f	부동소수(floating-point)
%s	문자열(String)	%o	8진수
%c	문자 1개(character)	%x	16진수
%d	정수(Integer)	%%	Literal % (문자 % 자체)

03 리스트(list)

● 리스트 만들기

```
리스트명 = [요소1, 요소2, 요소3, ...]
```

빈 리스트를 만들때 리스트명 = []


● 리스트 특징

```
>>> mylist=[1, 2, '해달이', True, ['a', 'b', 'c']]
>>> mylist
[1, 2, '해달이', True, ['a', 'b', 'c']]
```

요소의 자료형을 통일해주지 않아도 된다.

● 인덱싱(indexing)

```
>>> a = [1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

 a[0]	a[1]	a[2]	a[3]	a[4]	a[5]	a[6]	a[7]	a[8]	a[9]
1	2	3	4	5	6	7	8	9	10

a[-10]	a[-9]	a[-8]	a[-7]	a[-6]	a[-5]	a[-4]	a[-3]	a[-2]	a[-1]
1	2	3	4	5	6	7	8	9	10

03 리스트(list)

- 슬라이싱(slicing)

```
>>> a[0:3]
[1, 2, 3]
>>> a[1:3]
[2, 3]
>>> a[:3]
[1, 2, 3]
>>> a[7:]
[8, 9, 10]
>>> a[:]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10]
>>> a[-4:-2]
[7, 8]
```

a[x:y]라고 쓰면 a[x] 부터 a[y]전까지

- 리스트 수정

```
>>> a[0] = 100
>>> a
[100, 2, 3, 4, 5, 6, 7, 8, 9, 10]
```

리스트 인덱싱으로 직접 접근해서 값을 수정

- 리스트 삭제

```
>>> del a[0]
>>> a
[2, 3, 4, 5, 6, 7, 8, 9, 10]
```

```
>>> del a[3:]
>>> a
[2, 3, 4]
```

03 리스트(list)

- 메소드란?

내장 함수(built-in-function)와는 달리 특정 자료형이 가지고 있는 함수를 메소드(method)라고 한다.

- len() : 길이 구하기
- append() : 요소 추가
- sort() : 요소 정렬



파이썬 리스트의 내장함수 및 메소드 (Python List built-in functions and methods)

Built-in functions for List

- len(list) : 리스트 길이
- max(list) : 최대 요소
- min(list) : 최소 요소
- list(seq) : 리스트로 변환
- cmp(list1, list2) : 리스트 간 비교 (only for python 2.x version)

List methods

- list.append(obj) : 요소 추가
- list.extend(seq) : 리스트 추가
- list.count(obj) : 요소 개수
- list.index(obj) : 요소 위치 index
- list.insert(index, obj) : index 위치에 obj 삽입
- list.pop(obj=list[-1]) : 마지막 요소 제거
- list.remove(obj) : obj 객체 제거
- list.reverse() : 거꾸로 뒤집기
- list.sort(obj) : 정렬 (디폴트 오름차순)

R, Python 분석과 프로그래밍의 친구 <http://rfriend.tistory.com>

04 튜플(tuple)

- 튜플 생성

```
>>> t1 = ()
>>> t2 = (1,)
>>> t3 = (1, 2, 3)
>>> t4 = 1, 2, 3
>>> t5 = ('a', 'b', ('ab', 'cd'))
```

- 인덱싱, 슬라이싱

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0]
1
>>> t1[3]
'b'
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[1:]
(2, 'a', 'b')
```

- 튜플 더하기, 곱하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> t2 = (3, 4)
>>> t3 = t1 + t2
>>> t3
(1, 2, 'a', 'b', 3, 4)
```

```
>>> t2 = (3, 4)
>>> t3 = t2 * 3
>>> t3
(3, 4, 3, 4, 3, 4)
```

- 튜플 길이 구하기

```
>>> t1 = (1, 2, 'a', 'b')
>>> len(t1)
4
```


04 튜플(tuple)

- 튜플 특징

리스트와 달리 튜플은 요솟값을 한 번 정하면 지우거나 변경할 수 없다.

→ sort, insert, remove, pop과 같은 내장 함수가 없다.

```
>>> t1 = (1, 2, 'a', 'b')
>>> del t1[0]
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object doesn't support item deletion
```

```
>>> t1 = (1, 2, 'a', 'b')
>>> t1[0] = 'c'
```

```
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object does not support item assignment
```

05 딕셔너리(Dictionary)

- 딕셔너리 만들기

딕셔너리명 = {key1:Value1, Key2:Value2, Key3:Value3, ... }

- 예시

```
>>> dic = {'name':'pey', 'phone':'010-9999-1234', 'birth': '1118'}
```

key	value
name	pey
phone	010-9999-1234
birth	1118

List vs. Dictionary

List		Dictionary	
0	Element 0	Key 0	Value 0
1	Element 1	Key 1	Value 1
2	Element 2	Key 2	Value 2
⋮	⋮	⋮	⋮
Integer index	Element	Custom index	Element

05 딕셔너리(Dictionary)

- 딕셔너리 쌍 추가하기

```
>>> a = {1: 'a'}
>>> a[2] = 'b'
>>> a
{1: 'a', 2: 'b'}
```

- 딕셔너리 요소 삭제하기

```
>>> del a[1]
>>> a
{2: 'b', 'name': 'pey', 3: [1, 2, 3]}
```

- Key 사용해 Value 얻기

```
>>> a = {1: 'a', 2: 'b'}
>>> a[1]
'a'
>>> a[2]
'b'
```

```
>>> a = {'a': 1, 'b': 2}
>>> a['a']
1
>>> a['b']
2
```

- 딕셔너리 만들 때 주의사항

- Key는 고유한 값이므로 같은 Key를 가진 쌍이 존재할 수 없다.
- Key로 리스트를 사용할 수 없다. 튜플은 사용할 수 있다. 리스트는 수정될 수 있기 때문이다. 단, Value에는 변하는 값이든 변하지 않는 값이든 상관없이 아무 값이나 넣을 수 있다.

05 딕셔너리(Dictionary)

- 딕셔너리 관련 함수들
 - keys() : Key 리스트 생성
 - values() : Value 리스트 생성
 - items() : 쌍을 튜플로 묶은 값을 돌려줌
 - clear() : 모든 요소 삭제
 - get() : Key로 Value 얻기
 - in() : 해당 Key가 딕셔너리 안에 있는지 조사

06 집합(set)

- set 생성

```
>>> s1 = set([1,2,3])
>>> s1
{1, 2, 3}
```

```
>>> s2 = set("Hello")
>>> s2
{'e', 'H', 'l', 'o'}
```

set()의 괄호 안에 리스트를 입력하여 만들거나
문자열을 입력하여 만들 수도 있다.
비어 있는 집합 자료형은 s = set()

- set의 특징

- 중복을 허용하지 않는다.
- 순서가 없다(Unordered).

06 집합(set)

- 교집합, 합집합, 차집합 구하기

```
>>> s1 = set([1, 2, 3, 4, 5, 6])  
>>> s2 = set([4, 5, 6, 7, 8, 9])
```

- 교집합

```
>>> s1 & s2  
{4, 5, 6}
```

```
>>> s1.intersection(s2)  
{4, 5, 6}
```

- 합집합

```
>>> s1 | s2  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

```
>>> s1.union(s2)  
{1, 2, 3, 4, 5, 6, 7, 8, 9}
```

- 차집합

```
>>> s1 - s2  
{1, 2, 3}  
>>> s2 - s1  
{8, 9, 7}
```

```
>>> s1.difference(s2)  
{1, 2, 3}  
>>> s2.difference(s1)  
{8, 9, 7}
```

06 집합(set)

- set 관련 함수들
 - add : 값 1개 추가하기
 - update : 값 여러 개 추가하기
 - remove : 특정 값 제거하기

07 불 자료형(bool)

- 불(bool)

참(True)과 거짓(False)을 나타내는 자료형이다. 불 자료형은 다음 2가지 값만을 가질 수 있다.

- True - 참

- False - 거짓

```
>>> a = True
>>> b = False
```

```
>>> type(a)
<class 'bool'>
>>> type(b)
<class 'bool'>
```

type(x)는 x의 자료형을 확인하는 파이썬의 내장 함수

- 불 자료형은 조건문의 리턴 값으로도 사용된다.

```
>>> 1 == 1    >>> 2 > 1    >>> 2 < 1
True          True        False
```


07 불 자료형(bool)

- 자료형의 참과 거짓

값	참 or 거짓								
"python"	참	[1, 2, 3]	참	(1, 2, 3)	참	{'a': 1}	참	1	참
""	거짓	[]	거짓	()	거짓	{}	거짓	0	거짓
								None	거짓

문자열, 리스트, 튜플, 딕셔너리 등의 값이 비어 있으면(" ", [], (), {}) 거짓이 된다. 당연히 비어있지 않으면 참이 된다. 숫자에서는 그 값이 0일 때 거짓이 된다.

07 불 자료형(bool)

- 자료형의 참과 거짓 활용 방법

예시는 a가 참인 경우에 a.pop()을 계속 실행하여 출력하라는 의미이다.

a.pop() 함수는 리스트 a의 마지막 요소를 끄집어내는 함수이므로 리스트 안에 요소가 존재하는 한(a가 참인 동안) 마지막 요소를 계속해서 끄집어낼 것이다.

결국 더 이상 끄집어낼 것이 없으면 a가 빈 리스트([])가 되어 거짓이 된다. 따라서 while문에서 조건문이 거짓이 되므로 while문을 빠져나가게 된다.

```
>>> a = [1, 2, 3, 4]
>>> while a:
...     print(a.pop())
...
4
3
2
1
```



변수

자료형의 값을 저장하는 공간



01 변수

- 변수 : 자료형을 저장하는 공간, 자료를 담는 상자
파이썬에서 사용하는 변수는 객체를 가리키는 것이라고도 말할 수 있다.
객체란 우리가 지금껏 보아 온 자료형의 데이터(값)와 같은 것을 의미하는 말이다

변수 이름 = 변수에 저장할 값

변수를 만들 때는 =(assignment) 기호를 사용한다.

01 변수

- 변수 명명 규칙

변수 이름		변수 이름 짓기 규칙
nametag = "김코딩"	○	알파벳을 사용할 수 있습니다.
Nametag = "박써니"	○	대/소문자를 구분합니다.
키 = 160.2	○	한글을 사용할 수 있습니다.
name tag = "이소엔"	×	띄어쓰기는 할 수 없습니다.
name_tag = "이소엔"	○	밑줄 기호를 사용할 수 있습니다.
1234 = "박써니"	×	숫자만 사용할 수 없습니다.
1234name = "박써니"	×	숫자가 맨 앞에 올 수 없습니다.
nametag1 = "김코딩"	○	맨 앞 글자가 문자면 숫자를 사용할 수 있습니다.
print = "김코딩"	×	명령어는 변수로 사용하지 않는 게 좋습니다.

Check Point!



입출력

입력과 출력



01 입력

- `input()` : 사용자가 키보드로 입력한 모든 것을 '문자열'로 저장한다.

```
>>> a = input()
Life is too short, you need python
>>> a
'Life is too short, you need python'
>>>
```

```
>>> number = input("숫자를 입력하세요: ")
숫자를 입력하세요: 3
>>> print(number)
3
>>>
```

만약 정수형으로 입력받고 싶다면 `num = int(input())` 또는 `num_str = input()`
`num_int = int(num_str)`

02 출력

- `print()` : 데이터를 출력한다.
- 큰따옴표(")로 둘러싸인 문자열은 + 연산과 동일하다

```
>>> print("life" "is" "too short") # 1
lifeistoo short
>>> print("life"+"is"+"too short") # 2
lifeistoo short
```

- 문자열 띄어쓰기는 콤마로 한다.

```
>>> print("life", "is", "too short")
life is too short
```

- 한 줄에 이어서 출력하기

```
>>> for i in range(10):
...     print(i, end=' ')
...
0 1 2 3 4 5 6 7 8 9
```




제어문

조건문(if문), 반복문(for, while)



01 조건문(if문)

- if문 : 주어진 조건을 판단하여 해당 조건에 맞는 상황을 수행한다.

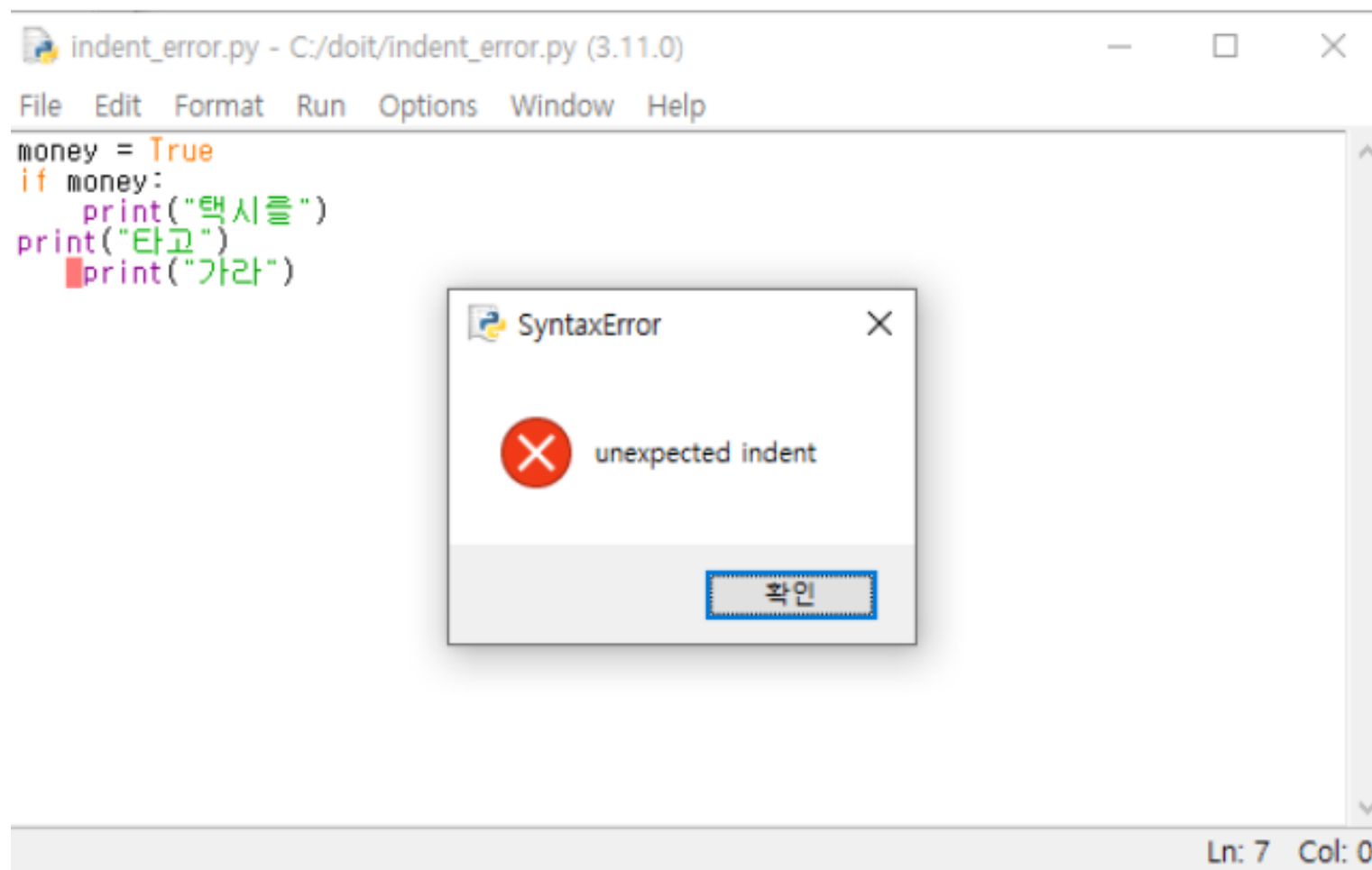
```
if 조건문:  
    수행할 문장1  
    수행할 문장2  
    ...  
else:  
    수행할 문장A  
    수행할 문장B  
    ...
```

```
>>> money = True  
>>> if money:  
...     print("택시를 타고 가라")  
... else:  
...     print("걸어 가라")  
...  
택시를 타고 가라
```

※ 조건문 다음에 콜론(:)을 잊지 말자!

01 조건문(if문)

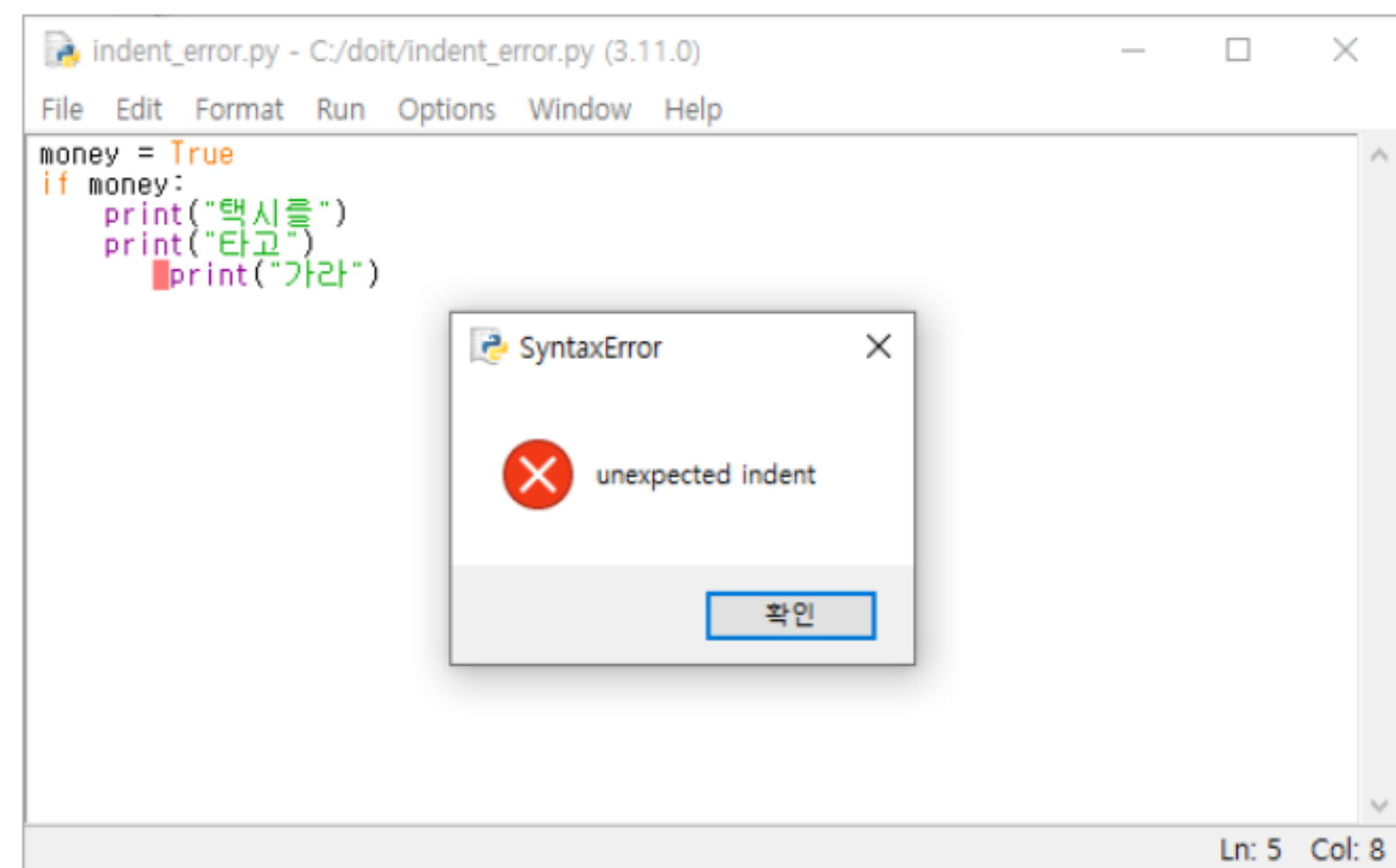
- 들여쓰기(indentation) : tab 한 번 또는 spacebar 4번



The screenshot shows a Python IDE window titled 'indent_error.py - C:/doit/indent_error.py (3.11.0)'. The code in the editor is:

```
money = True
if money:
    print("택시를")
    print("타고")
    print("가라")
```

A 'SyntaxError' dialog box is displayed in the foreground with the message 'unexpected indent' and a red 'X' icon. The dialog has a button labeled '확인' (Confirm). The status bar at the bottom right shows 'Ln: 7 Col: 0'.



The screenshot shows a Python IDE window titled 'indent_error.py - C:/doit/indent_error.py (3.11.0)'. The code in the editor is:

```
money = True
if money:
    print("택시를")
    print("타고")
    print("가라")
```

A 'SyntaxError' dialog box is displayed in the foreground with the message 'unexpected indent' and a red 'X' icon. The dialog has a button labeled '확인' (Confirm). The status bar at the bottom right shows 'Ln: 5 Col: 8'.

01 조건문(if문)

- 비교연산자(<, >, ==, !=, >=, <=)

```
>>> money = 2000
>>> if money >= 3000:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
걸어가라
>>>
```

비교연산자	설명
x < y	x가 y보다 작다
x > y	x가 y보다 크다
x == y	x와 y가 같다
x != y	x와 y가 같지 않다
x >= y	x가 y보다 크거나 같다
x <= y	x가 y보다 작거나 같다

01 조건문(if문)

- and, or, not

```
>>> money = 2000
>>> card = True
>>> if money >= 3000 or card:
...     print("택시를 타고 가라")
... else:
...     print("걸어가라")
...
택시를 타고 가라
>>>
```

연산자	설명
x or y	x와 y 둘중에 하나만 참이어도 참이다
x and y	x와 y 모두 참이어야 참이다
not x	x가 거짓이면 참이다

01 조건문(if문)

- 다양한 조건을 판단하는 elif

```
>>> pocket = ['paper', 'handphone']
>>> card = True
>>> if 'money' in pocket:
...     print("택시를 타고가라")
... else:
...     if card:
...         print("택시를 타고가라")
...     else:
...         print("걸어가라")
...
택시를 타고가라
>>>
```

```
>>> pocket = ['paper', 'cellphone']
>>> card = True
>>> if 'money' in pocket:
...     print("택시를 타고가라")
... elif card:
...     print("택시를 타고가라")
... else:
...     print("걸어가라")
...
택시를 타고가라
```

```
if <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
elif <조건문>:
    <수행할 문장1>
    <수행할 문장2>
    ...
else:
    <수행할 문장1>
    <수행할 문장2>
    ...
```

02 반복문(while문)

- while문 : 조건문이 참인 동안에 while문에 속한 문장들이 반복해서 수행된다.
- 조건과 상관없이 중간에 강제로 빠져나가고 싶다면 break를 사용한다.

```
while <조건문>:  
    <수행할 문장1>  
    <수행할 문장2>  
    <수행할 문장3>  
    ...
```

```
>>> coffee = 10  
>>> money = 300  
>>> while money:  
...     print("돈을 받았으니 커피를 줍니다.")  
...     coffee = coffee - 1  
...     print("남은 커피의 양은 %d개입니다." % coffee)  
...     if coffee == 0:  
...         print("커피가 다 떨어졌습니다. 판매를 중지합니다.")  
...         break  
...  
...
```

02 반복문(while문)

```
>>> treeHit = 0
>>> while treeHit < 10:
...     treeHit = treeHit +1
...     print("나무를 %d번 찍었습니다." % treeHit)
...     if treeHit == 10:
...         print("나무 넘어갑니다.")
...
```

나무를 1번 찍었습니다.
나무를 2번 찍었습니다.
나무를 3번 찍었습니다.
나무를 4번 찍었습니다.
나무를 5번 찍었습니다.
나무를 6번 찍었습니다.
나무를 7번 찍었습니다.
나무를 8번 찍었습니다.
나무를 9번 찍었습니다.
나무를 10번 찍었습니다.
나무 넘어갑니다.

treeHit	조건문	조건판단	수행하는 문장	while문
0	0 < 10	참	나무를 1번 찍었습니다.	반복
1	1 < 10	참	나무를 2번 찍었습니다.	반복
2	2 < 10	참	나무를 3번 찍었습니다.	반복
3	3 < 10	참	나무를 4번 찍었습니다.	반복
4	4 < 10	참	나무를 5번 찍었습니다.	반복
5	5 < 10	참	나무를 6번 찍었습니다.	반복
6	6 < 10	참	나무를 7번 찍었습니다.	반복
7	7 < 10	참	나무를 8번 찍었습니다.	반복
8	8 < 10	참	나무를 9번 찍었습니다.	반복
9	9 < 10	참	나무를 10번 찍었습니다. 나무 넘어갑니다.	반복
10	10 < 10	거짓		종료

03 반복문(for문)

- for문의 기본구조

```
for 변수 in 리스트(또는 튜플, 문자열):  
    수행할 문장1  
    수행할 문장2  
    ...
```

```
>>> test_list = ['one', 'two', 'three']  
>>> for i in test_list:  
...     print(i)  
...  
one  
two  
three
```

- for문의 응용

```
# marks1.py  
marks = [90, 25, 67, 45, 80]  
  
number = 0  
for mark in marks:  
    number = number + 1  
    if mark >= 60:  
        print("%d번 학생은 합격입니다." % number)  
    else:  
        print("%d번 학생은 불합격입니다." % number)
```

03 반복문(for문)

- for문과 continue

for문 안의 문장을 수행하는 도중에 continue문을 만나면 for문의 처음으로 돌아가게 된다.

```
# marks2.py
marks = [90, 25, 67, 45, 80]

number = 0
for mark in marks:
    number = number + 1
    if mark < 60:
        continue
    print("%d번 학생 축하합니다. 합격입니다. " % number)
```

```
C:\doit>python marks2.py
1번 학생 축하합니다. 합격입니다.
3번 학생 축하합니다. 합격입니다.
5번 학생 축하합니다. 합격입니다.
```

03 반복문(for문)

- for문과 함께 자주 사용하는 range 함수

for문은 숫자 리스트를 자동으로 만들어 주는 range 함수와 함께 사용하는 경우가 많다.

```
>>> add = 0
>>> for i in range(1, 11):
...     add = add + i
...
>>> print(add)
55
```

range(1, 11)은 숫자 1부터 10까지(1 이상 11 미만)의 숫자를 데이터로 갖는 객체이다.

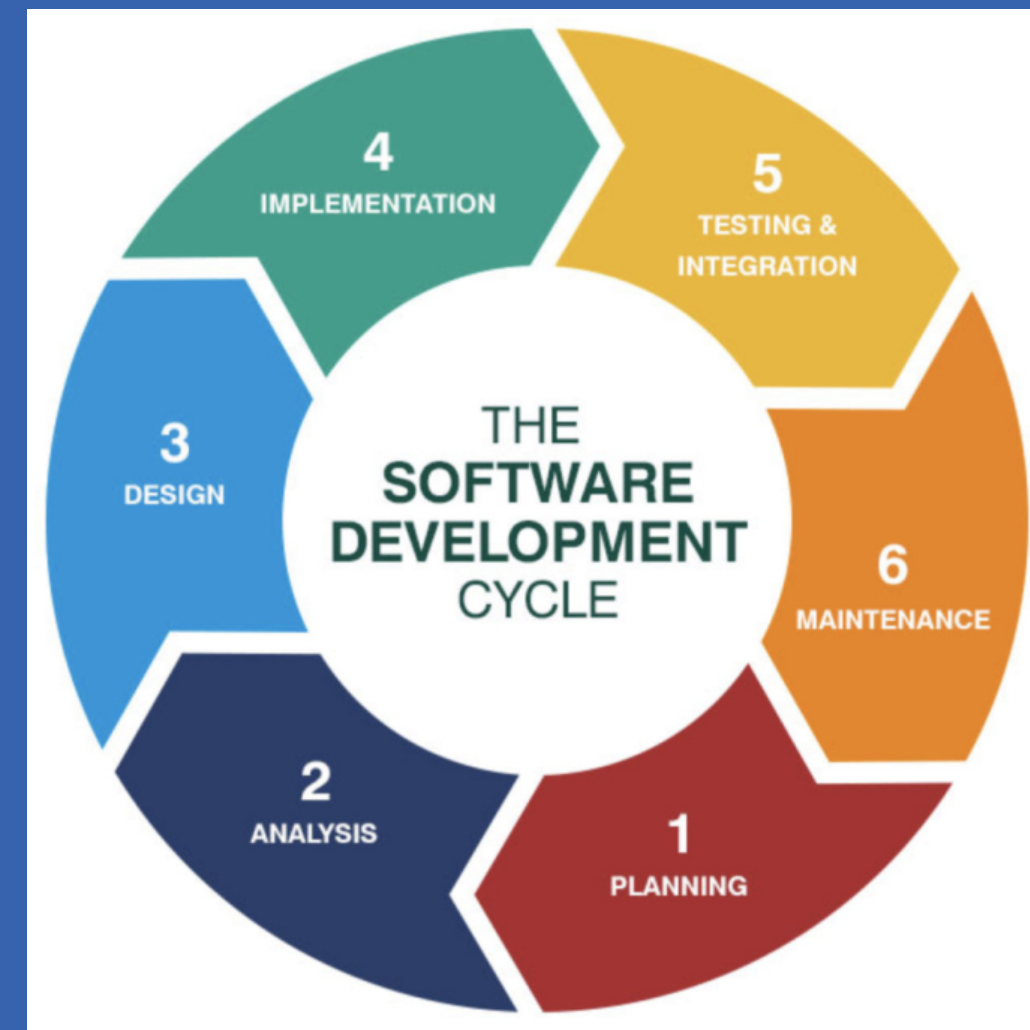
따라서 위 예에서 i 변수에 숫자가 1부터 10까지 하나씩 차례로 대입되면서 add = add + i 문장을 반복적으로 수행하고 add는 최종적으로 55가 된다.

- ※ range(11)은 0부터 11 미만의 숫자를 포함하는 range 객체를 만들어 준다.
- ※ range(a,b,c)에서 a는 시작값, b는 종료값, c는 간격이다. (ex)range(10,0,-1)

SDLC

Software Development Life Cycle

1. 분석 : 문제 파악
2. 디자인 : 데이터/ 알고리즘 정리
3. 프로그램 작성 : 실행
4. 테스트 : 오류 확인 및 수정





사전과제 설명

2번, 3번



2번 과제(01.py)

```
1 print("당신의 이름은 무엇입니까?")
2 name=input()
3 print("당신은 어떤 과일을 좋아합니까?")
4 fruit=input()
5 print(name+"씨가 좋아하는 과일은 "+fruit+"입니다!")
6
```

3번 과제(02.py)

```
1 n,x = map(int,input().split())
2 num = list(map(int,input().split()))
3
4 for i in range(n):
5     if num[i] > x:
6         print(num[i],end=' ')
7
```



세션 실습



노션 'Python(1) 세션 실습'의 문제들을 자유롭게 풀어보세요!
8시 50분까지 실습이 진행되고, 50분에 노션에 정답 코드를 올려드립니다.
구글링을 적극 활용해보세요!

수고하셨습니다!

- 본 세션 자료는 온라인에 무료로 나와있는 도서 '점프 투 파이썬(박응용)' 위키독스를 참고했습니다.
- 사전 과제 노션 페이지 하단에서 소개한 백준 사이트에서 문제를 많이 풀어보시길 바랍니다.