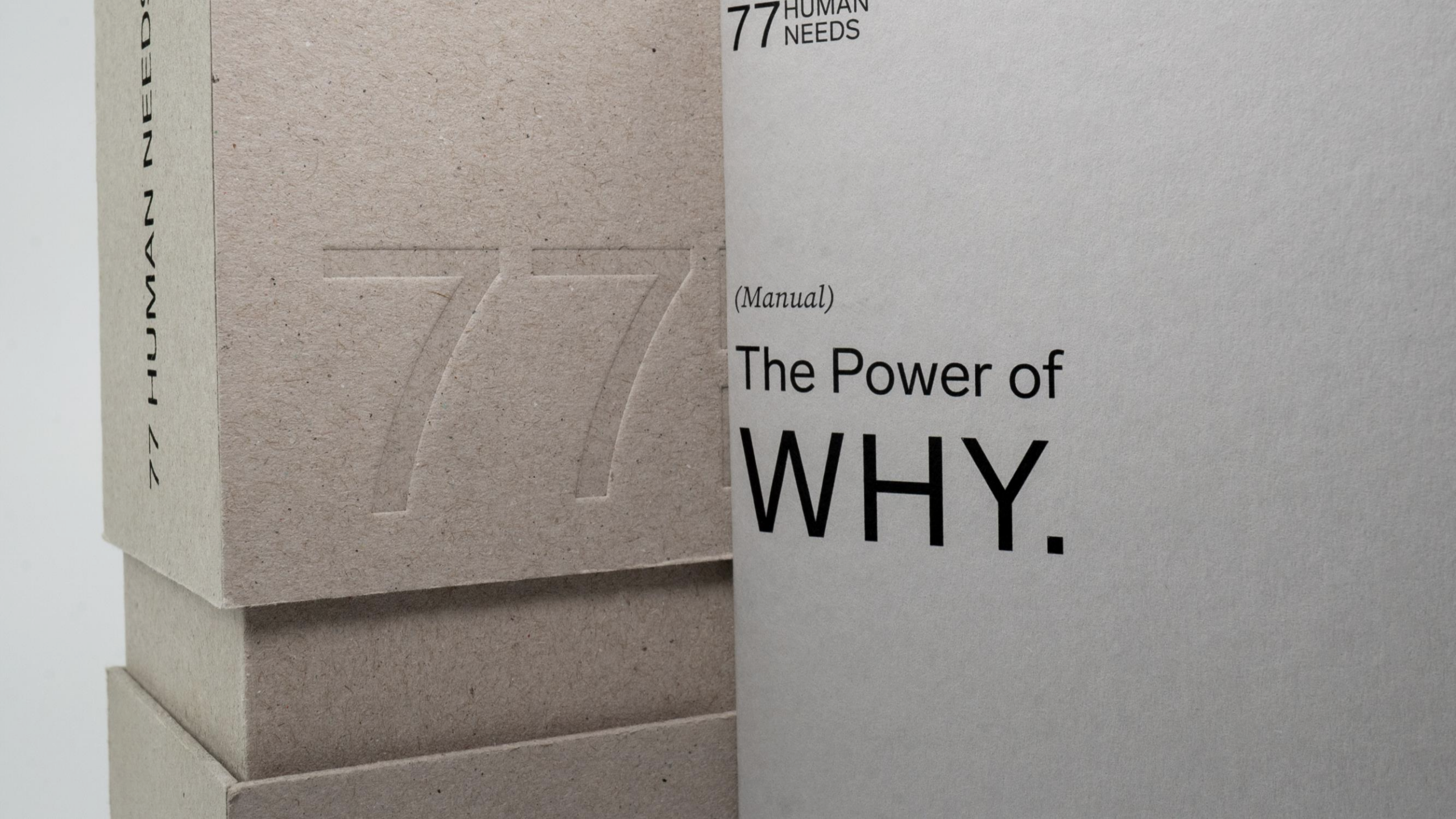
python

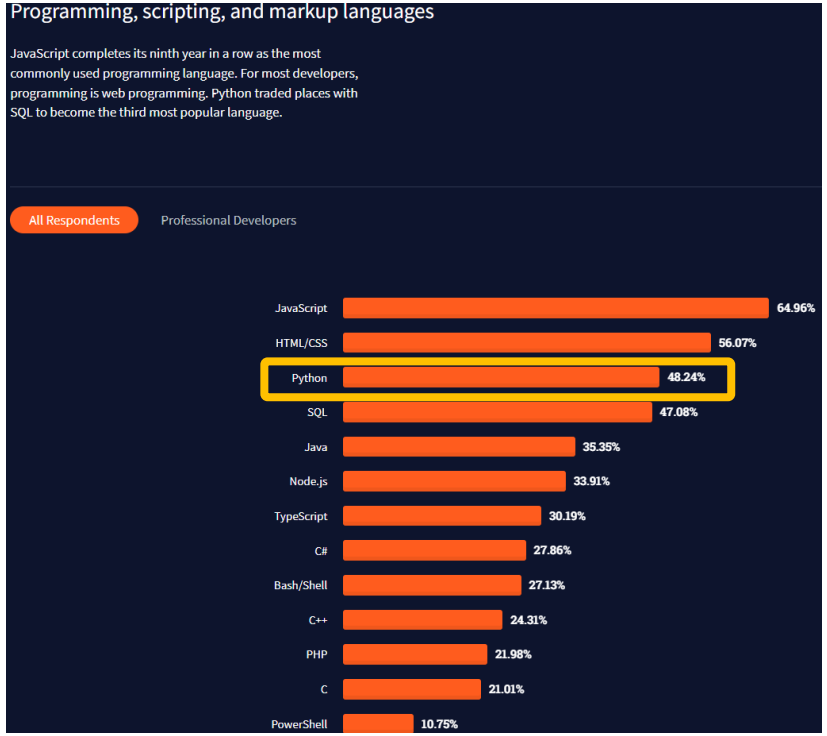
SOFTWARE FOUNDATION


77 HUMAN NEEDS
77
77 HUMAN NEEDS
(Manual)
The Power of
WHY.

Life is too short,

You need Python

- 프로그래밍이 간결하다
- 빠른 개발
- 사람 언어와 유사한 프로그래밍 언어



Instagram

YouTube

Dropbox

1991년 귀도 반 로섬이 발표한 고급 프로그래밍 언어

언어 이름은 귀도가 좋아하는 코미디 Monty Python`s Flying Circus에서 따옴

인터프리식 언어, 객체지향적, 동적 타이핑 대화형 언어

비영리 파이썬 소프트웨어 재단이 관리하고 있으며, 사실상 표준은 C언어로 구현된 Cython

Python2 2000년, Python3 2008년 / Python2 → 2020년 지원 중단

귀도 반 로섬

Over six years ago, in December 1989, I was looking for a "hobby" programming project that would keep me occupied during the week around Christmas. My office (a government-run research lab in Amsterdam) would be closed, but I had a home computer, and not much else on my hands. I decided to write an interpreter for the new scripting language I had been thinking about lately: a descendant of ABC that would appeal to Unix/C hackers. I chose Python as a working title for the project, being in a slightly irreverent mood (and a big fan of Monty Python's Flying Circus).

*인터프리식 언어: 코드를 한줄 한줄 읽어가며 명령을 처리

**객체지향적: 파이썬은 Object(객체)로 이루어져 있으며, 객체 지향 프로그래밍을 추구하고 있다.

OOP(Object Oriented Programming, 객체 지향 프로그래밍): 프로그래밍에 필요한 데이터를 추상화하여 상태와 행위를 가진 객체를 만들고,
객체들 간의 유기적 상호작용을 통해 로직을 구성하는 방법

***동적 타이핑 대화형 언어: 파이썬은 변수를 만들 때 변수의 타입을 먼저 지정해주지 않습니다. 그리고, 프로그램 자체가 영어 친화적으로 대화하듯 되어 되어 있습니다.

파이썬으로 할 수 있는 일

- 시스템 유틸리티 제작
- GUI 프로그래밍
- C/C++ 결합
- 웹 프로그래밍
- 수치 연산 프로그래밍
- 데이터베이스 프로그래밍
- 데이터 분석
- IoT 제어

파이썬으로 할 수 없는 일

- 시스템과 밀접한 프로그래밍 영역

  ex) OS
- 모바일 프로그래밍

IDE

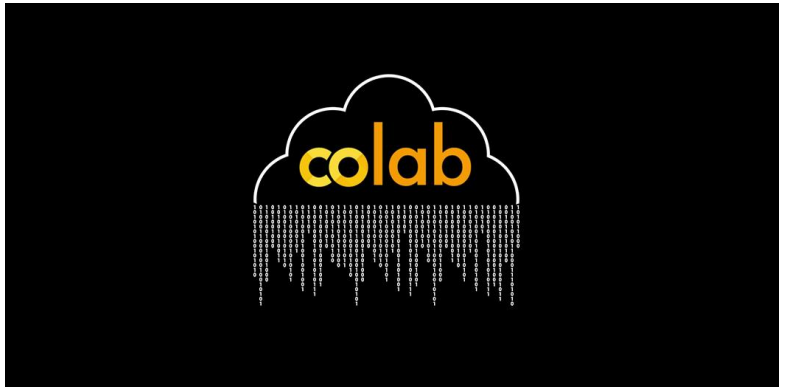
통합 개발 환경 / 코딩, 디버그, 컴파일 배포 등

프로그램 개발에 관련된 모든 작업을 하나의 프로그램 안에서 처리하는 환경을 제공하는 소프트웨어

에디터

VS Code, PyCharm

colab



코랩(colab)은 구글에서 제공하고 있는 클라우드 주피터 노트북입니다.

Jupyter Notebook이 무엇일까요?

주피터 노트북

노트북이라는 말 그대로 공책을 사용하듯 코드를 작성하면서 설명도 넣을 수 있습니다.

코랩을 사용하는 이유?

코랩은 클라우드 서비스이기에 내가 사용하는 PC의 CPU나 GPU가 낮아도 괜찮다라는 부분이 이점입니다.

로컬에 프로그램을 따로 설치할 필요가 없으며, 머신러닝과 같이 GPU를 많이 사용할 때 클라우드 서비스를 사용하는데

코랩은 이러한 점에서 인공지능을 배우기에 이점이 있습니다.

출력문
print

Print(''출력할 내용'')

파이썬에서는 작은 따옴표(')와 큰 따옴표(")를 구별하지 않습니다.

여러줄 작성시 docstring 형태로 사용합니다.

주석, Comment

설명문을 작성할 때 사용

주석을 많이 쓰는 것은 좋지 않음 → 코드만 보고 이해가 되게 프로그래밍 하는게 능력

#을 사용하거나 docstring 사용

Docstring:

https://www.python.org/dev/peps/pep-0257/

출력문
print

end keyword argument

print('Hello, Python', end='!')

end는 출력문이 끝나는 지점에 어떻게 끝낼 것인지 → default: ₩n

sep keyword argument

print('Hello', 'Python', sep=', ')

Sep은 출력문에서 `,`로 나누어진 부분 사이에 특정값을 넣는 방법 → default: white space
(한 칸 띄어쓰기)

변수
variable

변수

특정 값을 저장하는 메모리 공간

`num = 5` 라고 한다면 num이라는 변수가 5라는 값을 가지게 된다는 것을 알 수 있습니다.

이러한 식에서 아래와 같이 하게 된다면 수학적으로 보면 모순이 될 것입니다.

`num = num + 3`

수학적으로 바라보면 안되고, 프로그래밍적으로 보게 되면 num이라는 변수에 현재 가지고 있는 num의 값 + 3을 다시 값을 담아주는 과정이라고 보시면 됩니다.

이렇게 값을 정해주는 것을 **할당**이라고 표현할 것입니다.

변수
variable

변수 명명규칙

1. 영문자와 숫자, _만 사용할 수 있습니다.
   - 한글도 사용은 가능하나, 영어 사용을 권장합니다.
   - _ 이외의 특수문자는 사용할 수 없습니다.
2. 변수명에 공백이 들어가면 안됩니다.
3. 대소문자를 구별합니다.
4. 문자나 _부터 시작해야 합니다.
5. Python 키워드는 사용할 수 없습니다.

```
import keyword

print(keyword.kwlist)
```

['False', 'None', 'True', '__peg_parser__', 'and', 'as', 'assert', 'async', 'await', 'break', 'class', 'continue', 'def', 'del', 'elif', 'else', 'except', 'finally', 'for', 'from', 'global', 'if', 'import', 'in', 'is', 'lambda', 'nonlocal', 'not', 'or', 'pass', 'raise', 'return', 'try', 'while', 'with', 'yield']

변수
variable

f-string

f-string은 3.6버전 이상에서 지원하고 있는 문자 포맷팅 방식입니다.

따옴표 앞에 f를 쓰고 중괄호 안에 해당 변수를 사용하여 편리하게 문자 포맷을 진행할 수 있습니다.

print(3 + '안녕') 을 하게 되면 결과가 어떻게 나올까요?

확인을 하셨다면 이제는

print(f'{3}' + '안녕') 으로 확인을 해주세요.

type(3) 의 결과와 type(f'{3}') 의 결과를 비교해보면 이유를 알 수 있습니다.

type()

타입을 확인 할 수 있는 내장 함수입니다.

숫자
Number

Number

파이썬에서 숫자를 나타내는 기본 타입은 int와 float가 있습니다.

int

Integer의 의미를 가지고 있으며, 숫자의 정수형을 가지는 타입입니다.

여기서 정수는 음의 정수, 0, 양의 정수(=자연수)를 의미합니다.

float

소수점의 값을 가집니다.

Complex 복소수형은 자주 쓰지 않아 생략하였습니다.

숫자
Number

사칙 연산

숫자의 사칙 연산 +, -, x, /가 있는 것처럼 프로그래밍에서는 아래와 같습니다.

수학	Python
+(더하기)	+
-(빼기)	-
x(곱하기)	*
/(나누기)	/
	//

Python 나누기에 /는 소수점이 나오게 되고, //는 몫만 나오게 되는 나누기입니다.

%을 이용하면 나눗셈의 나머지를 구할 수 있습니다.

숫자
Number

1. 3을 5번 곱한 값을 작성해주세요.

2. 1.2 - 0.1의 값을 구해주세요.

문자열
String

string

문자열 데이터 타입

문자열도 동일하게 연산이 가능합니다. (+ 와 * 연산)

Escape code

코드	설명
₩n	문자열 안에서 줄 바꿈
₩t	문자열 안에서 탭(띄어쓰기) 사용
₩'	문자 ' 를 쓰고자 할 때 사용
₩''	문자 ''를 쓰고자 할 때 사용
₩r	캐리지 리턴(줄 바꿈 문자, 현재 커서를 가장 앞으로 이동)
₩f	폼 피드(줄 바꿈 문자, 현재 커서를 다음 줄로 이동)
₩b	백 스페이스

문자열
String

인덱싱 / 슬라이싱

문자열은 반복가능한 객체로 인덱싱과 슬라이싱이 가능합니다.

인덱싱이란 데이터 검색에서 순서를 통해 빠르게 가져올 수 있는 방법입니다.

슬라이싱이란 인덱스를 통해서 시작 위치에서 끝나는 지점까지 잘라서 값들을 가져오는 방법입니다.

data =	P	y	t	h	o	n
INDEX	0	1	2	3	4	5
Reverse INDEX	-6	-5	-4	-3	-2	-1

인덱스(index)는 0부터 시작합니다.

Python이라는 문자열을 data변수에 담아서 data[0]이라고 하면 P라는 값이 나오게 됩니다.

슬라이싱 → data[초기시작index : 끝index, 증감정도]

data[2:4]라고 하면 th라는 값을 얻을 수 있습니다. 슬라이싱의 마지막 인덱싱 값은 포함하지 않습니다.

문자열
String

len

반복 가능한 객체들은 크기를 구할 수 있습니다.

len(iterable Object)

cf) 파이썬의 자료형은 모두 객체로 이루어져 있습니다.

논리형
Bool

bool

논리형 boolean type으로 True / False 2가지 값만 가집니다.

조건식의 결과는 참과 거짓으로 나뉘어집니다.

비어있는 경우 False 입니다.

''	''''	[]	list()	0
()	tuple()	{}	set()	dict()
None	False	...		

캐스팅
Casting

Casting

데이터 타입을 강제적으로 변환해줄 수 있습니다.

변환해줄 타입(데이터)

ex)

```
data = 5
data = str(data) #문자열로 변환
```

단, 모든 경우 성립은 아닙니다.

int('ab') 이러한 경우처럼 자료형이 작은 경우 → 큰 경우는 가능하지만,

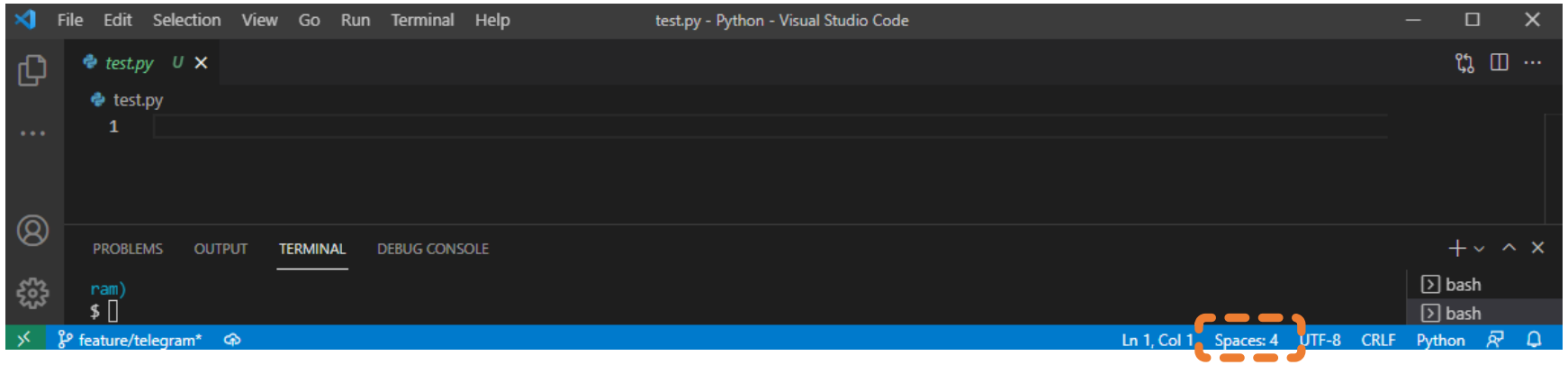
큰 자료형 → 작은 자료형은 문제가 생깁니다.

반복문
Loop

Intent

파이썬 구문의 영역을 지정할 때는 들여쓰기(intent)를 사용합니다.

PEP8에서 space 4번을 기준으로 합니다.



반복문
Loop

For문

```
for 변수 in iterable객체:
    print(변수)
```

Iterable객체들을 하나씩 확인하는 방법도 있고

```
for 변수 in range(초기값, end값, 증감정도):
    print(변수)
```

range 자체가 iterable하기에 범위 함수를 통해서 사용할 수도 있습니다.

range

1. range(end)
2. range(start, end, step)

(단, end는 포함하지 않습니다.)

반복문
Loop

1. 반복문을 통해서 구구단을 만들어 보세요.

반복문
Loop

While문

For문에서는 지정된 숫자만큼 반복을 하였는데 계속 반복하거나

조건식이 참일 때만 반복하기 위해서 사용하는 Loop문입니다.

```
while (조건식):
    print(값)
```

혹여나 실수로 `while True`와 같이 빠져나갈 수 없게 프로그래밍이 가동되었다면,

Ctrl + c 를 통해 빠져나올 수 있습니다.

반복문
Loop

Continue

반복문을 진행하다가 특정한 값이 나오면 넘어가게 하고 싶을 때 continue을 사용합니다.

Break

반복문을 진행하다가 특정한 값이 나와 반복문을 멈추고 싶을 때 break을 사용합니다.

Enumerate

열거하다라는 뜻을 가지고 있으며, Iterable 객체를 입력받아 인덱스 값을 포함하는 enumerate객체로 반환합니다.

가정문
if

If문

가정문으로 조건식이 참이면 수행문장 실행 / 거짓이면 실행하지 않습니다.

```
if (조건식):
    수행문장
```

If~else문

```
if (조건식):
    수행문장A
else:
    수행문장B
```

조건이 참일 때는 수행문장A 실행

거짓일 경우 else문의 수행문장B 실행

If~elif~else문

```
if (조건식A):
    수행문장A
elif (조건식B):
    수행문장B
else:
    수행문장C
```

조건A가 참일 때는 수행문장A 실행

조건A가 거짓일 때는

조건B가 참이라면 수행문장B 실행

조건B가 거짓이면 else문의 수행문장C 실행

가정문
if

Pass

조건문 안에 들어왔는데 아무것도 하지 않을 때 사용합니다.

조건문에 들어는 왔지만 아무것도 하지 않을 때 pass을 사용하여 수행하지 않고 넘어갈 수 있습니다.

프로그램 작성시 if문 안에 수행문장이 생각나지 않는데 이후의 과정을 작성하거나 중간에 확인할 때

```
if 조건식:
    pass
```

이렇게 두고 작업하기도 합니다.

가정문
if

1. 베스킨31 게임을 진행합니다. 1부터 31까지 출력하며 3, 6, 9가 들어가면 `박수`를 출력해주세요.

가정문
if

삼항연산자

If~else문을 간단한 수행문장이라면 한 줄로 처리할 수 있습니다.

```
if (조건식):
    수행문장A
else:
    수행문장B
```

참일때 값 if 조건식 else 거짓일때 값

리스트
List

디버깅 편리한 참조 사이트

https://pythontutor.com

List

순서가 있는 수정가능한 객체의 집합 입니다.

String 타입과 동일하게 +와 * 연산이 가능합니다.

리스트도 인덱싱과 슬라이싱이 동일하게 적용 됩니다.

리스트 생성방법 1. list()

2. []

Mutable

리스트는 수정이 가능(Mutable)하기에 인덱스로 접근하여 해당 값만을 변경할 수 있습니다.

del()

del함수를 통해 해당 리스트 요소나 객체를 삭제할 수 있습니다.

리스트
List

remove()

remove함수를 통하여 해당 값을 삭제할 수 있습니다.

List객체.remove(값) → 해당 값에서 가장 앞에 나와 있는 요소를 삭제합니다.

pop()

리스트에서 값을 꺼내옴으로서, 해당 값을 지우는 효과를 낼 수 있습니다.

꺼내온 값을 반환 받습니다.

pop_data = List객체.pop(index값)

(index값 default → -1)

Stack / Queue

Stack: FILO

Queue: FIFO

리스트
List

append()

리스트에 값을 추가하는 함수

List객체.append(해당값) → 넣어진 값은 가장 뒤에 추가됩니다.

insert()

리스트에 값을 삽입하는 함수

append함수는 리스트 가장 뒤에 추가해주는데 가장 앞에 두거나 중간값에 넣어야할 때 insert함수를 사용합니다.

List객체.insert(index, 값) → index에 위치를 적어주고, 해당 값을 넣어 해당 위치에 추가

(해당 위치 뒤에 있는 값들은 인덱스가 +1씩 되어집니다.)

리스트
List

sort()

List객체.sort() → 리스트의 요소를 정렬해 줍니다.

sort()와 sorted()의 차이점

List객체.sort()

sorted(List객체)

reverse()

List객체.reverse() → 리스트를 역순으로 뒤집어 줍니다.

리스트
List

index()

List객체.index(값) → 리스트 요소에서 값의 인덱스를 반환해줍니다.

동일한 값이 여러 개일 경우에는 가장 앞에 있는 인덱스가 반환됩니다.

extend()

List객체A.extend(list객체B) → 리스트를 확장해주는 역할을 합니다.

쉽게 list + list의 결과를 나타낸다고 보면 됩니다.

리스트
List

1. person = ['A', 'B', 'A', 'A', 'AB', 'O', 'AB', 'B', 'A', 'AB'] 에서
   A형은 몇 명인지, B형은 몇 명인지, O형은 몇 명인지, AB형은 몇 명인지 출력해주세요.

2. 학생들이 성적이 다음과 같을 때 점수가 가장 잘 나온 학생이 몇 점인지 출력해주세요.
   list_data = ['최웅', 77, '국연수', 93, '김지웅', 91, '엔제이', 88, '이솔이', 75]

리스트
List

3. 아래 모양과 같이 이중 리스트를 만들어주세요.

이중 리스트의 초기는 0으로 채워주시면 됩니다.

list_data = [[0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0], [0,0,0,0,0]]

1	2	3	4	5
6	7	8	9	10
11	12	13	14	15
16	17	18	19	20
21	22	23	24	25

```
list_data = [[0] * 5 for _ in range(5)]
```

이후에 다룰 리스트 컴프리헨션

리스트
List

4. 아래 모양과 같이 이중 리스트를 만들어주세요.

1	2	3	4	5
10	9	8	7	6
11	12	13	14	15
20	19	18	17	16
21	22	23	24	25

리스트
List

5. 아래 모양과 같이 이중 리스트를 만들어주세요.

1	2	3	4	5
16	17	18	19	6
15	24	25	20	7
14	23	22	21	8
13	12	11	10	9

튜플
Tuple

Tuple

리스트와 거의 비슷하나, 수정 불가능(immutable) 객체이다.

String 타입과 동일하게 +와 * 연산이 가능합니다.

튜플도 인덱싱과 슬라이싱이 동일하게 적용 됩니다.

튜플 생성방법 1. tuple()

2. ()

Packing / Unpacking

패킹: 포장

언패킹: 언박싱

Packing: 하나의 변수에 여러 개의 값을 넣는 것

Unpacking: 패킹된 변수에서 여러 개의 값을 꺼내는 것

사전형
Dictionary

Dict

딕셔너리는 해시(Hash) 기법을 이용해서 데이터를 저장합니다.

보통 딕셔너리와 같은 **키-값** 형태의 자료형을 해시, 해시 맵, 해시 테이블 등으로 부르기도 합니다.

딕셔너리를 만드는 방법

1. dict()
2. {}

   ({}는 이후에 배울 집합형에서도 사용됩니다)

딕셔너리 key값은 immutable 객체가 와야하며, value는 mutable합니다.

사전형
Dictionary

추가 / 삭제

- 추가: 할당과 동일 (값이 있을시 변경됨)
- 삭제: del 함수를 통해서 요소/객체 삭제

keys()

Dictionary객체.keys() → 딕셔너리의 키 값들을 반환합니다.

values()

Dictionary객체.values() → 딕셔너리의 값들을 반환합니다.

items()

Dictionary객체.items() → 딕셔너리의 키와 값들을 반환합니다.

사전형
Dictionary

clear()

Dictionary객체.clear() → 해당 딕셔너리 객체를 빈 딕셔너리로 만듭니다.

get()

Dictionary객체.get(키값) → 키값이 있으면 값 반환 / 없다면 None 반환

인덱싱으로 접근할 때 있으면 값을 반환하지만 없다면 에러가 발생하여 이러한 경우를 방지하고자 get함수 사용

연산자 in

구성원 연산자 in은 해당 값이 iterable객체에 있는지 판단해줍니다.

해당값 in iterable객체 → 있다면 True / 없다면 False

사전형
Dictionary

1. person = ['A', 'B', 'A', 'A', 'AB', 'O', 'AB', 'B', 'A', 'AB'] 에서

   A형은 몇 명인지, B형은 몇 명인지, O형은 몇 명인지, AB형은 몇 명인지 딕셔너리 형태로 만들어주세요.

   (주의! 직접 변수명을 만들어 작성하는 것이 아닌, person 변수를 이용해서 리스트 → 딕셔너리화 하는 것입니다.)

```
Ex) result = {
    'A': 4,
    'B': 2,
    ...
}
```

집합
Set

Set

집합에 관련된 데이터들을 쉽게 처리하기 위해 만들어진 자료형 입니다.

집합을 만드는 방법

1. set()
2. {}

집합의 특징

1) 중복을 허용하지 않습니다.

2) 순서가 없습니다. → 인덱싱을 허용하지 않음

3) 수정 가능(mutable) 객체입니다.

(단, 집합 내부 원소는 mutable 할 수 없습니다.)

집합
Set

합집합

1. | (or) 이용한 방법
2. Union 내장함수
   - 집합객체A.union(집합객체B)

차집합

1. - (빼기) 이용한 방법
2. difference 내장함수
   - 집합객체A.difference(집합객체B)

교집합

1. & (and) 이용한 방법
2. intersection 내장함수
   - 집합객체A.intersection(집합객체B)

대칭차집합

대칭차집합 = 합집합 - 교집합

1. ^ (xor) 이용한 방법
2. symmetric_difference 내장함수
   - 집합객체A. symmetric_difference(집합객체B)

집합
Set

add()

집합객체.add(값) → 집합 요소 추가하기

update()

집합객체.update(iterable객체) → 집합 여러 요소 추가하기

remove()

집합객체.remove(값) → 특정값 제거하기

집합
Set

issubset()

집합객체A.issubset(집합객체B) → 집합객체A가 집합객체B의 부분집합이라면 True / 아니면 False 반환

isuperset()

집합객체A.isuperset(집합객체B) → issubset과 반대의 의미

isdisjoint()

집합객체A.isdisjoint(집합객체B) → 교집합이 없다면 True / 있다면 False 반환

컴프리헨션
Comprehension

List Comprehension

list_data = [변수 for 변수 in iterable객체]

list_data = [변수 for 변수 in iterable객체 if 조건식]

list_data = [삼항연산자 for 변수 in iterable객체]

list_data = [[변수A for 변수A in iterable객체] for 변수B in iterable객체]

list_data = [[변수] * 숫자 for 변수 in iterable객체]

컴프리헨션
Comprehension

Set Comprehension

set_data = { 변수 for 변수 in iterable객체 }

set_data = { 변수 for 변수 in iterable객체 if 조건식 }

set_data = { 삼항연산자 for 변수 in iterable객체 }

컴프리헨션
Comprehension

Dict Comprehension

```
dict_data = { key: value for key, value in iterable객체 }

dict_data = { key: value for key, value in iterable객체 if 조건식 }

dict_data = { 삼항연산자 for key, value in iterable객체 }
```

연산자
Operator

산술연산자

연산자	설명	예시
+	더하기	a + b = 25
-	빼기	a - b = 5
*	곱하기	a * b = 150
/	나누기	a / b = 1.5
//	몫	a // b = 1
%	나머지	a % b = 5
**	제곱	a ** b = 255

연산자
Operator

비교연산자

연산자	설명	예시
==	값이 같다	a == b
!=	값이 같지 않음(값이 다르다)	a != b
>	왼쪽 값이 오른쪽 값보다 크다	a > b
<	왼쪽 값이 오른쪽 값보다 작다	a < b
>=	왼쪽 값이 오른쪽 값보다 크거나 같다	a >= b
<=	왼쪽 값이 오른쪽 값보다 작거나 같다	a <= b

연산자
Operator

할당연산자

연산자	설명	예시
=	왼쪽 변수에 오른쪽 값 할당	a = 3 + 5
+=	왼쪽 변수에 오른쪽 값 더하고 결과를 왼쪽 변수에 할당	a += 3
-=	왼쪽 변수에 오른쪽 값 빼고 결과를 왼쪽 변수에 할당	a -= 5
*=	왼쪽 변수에 오른쪽 값 곱하고 결과를 왼쪽 변수에 할당	a *= 3
/=	왼쪽 변수에서 오른쪽 값 나누고 결과를 왼쪽 변수에 할당	a /= 2
//=	왼쪽 변수에서 오른쪽 값 나눈 몫의 결과를 왼쪽 변수에 할당	a //= 5
%=	왼쪽 변수에서 오른쪽 값 나눈 나머지의 결과를 왼쪽 변수에 할당	a %= 7
**=	왼쪽 변수에서 오른쪽 값만큼 제곱을 하고 결과를 왼쪽 변수에 할당	a **=3

연산자
Operator

논리연산자

연산자	설명	예시
and	논리 AND 연산. 둘 다 참일 때만 참	a and b
or	논리 OR 연산. 둘 중 하나만 참이면 참	a or b
not	논리 NOT 연산. 논리 상태를 반전	not a

연산자
Operator

비트연산자

연산자	설명	예시
&	AND 연산, 둘 다 참일 때 만족	a & b
|	OR 연산, 둘 중 하나만 참이면 만족	a | b
^	XOR 연산, 둘 중 하나만 참일 때 만족	a ^ b
~	보수 연산	~a
<<	왼쪽 시프트 연산자. 변수의 값을 왼쪽으로 지정된 비트 수만큼 이동	a << 2
>>	오른쪽 시프트 연산자. 변수의 값을 오른쪽으로 지정된 비트 수만큼 이동	a >> 2

연산자
Operator

구성원 연산자

멤버쉽 연산자라고도 불리며, 해당 값이 컬렉션 속에 있는지 없는지 확인합니다.

연산자	설명	예시
in	list 내에 포함되어 있으면 참	a in list
not in	list 내에 포함되어 있지 않으면 참	a not in list

연산자
Operator

식별 연산자

연산자	설명	예시
is	개체 메모리 위치나 값이 같다면 참	a is b
is not	개체 메모리 위치나 값이 같지 않으면 참	a is not b

연산자
Operator

연산자 우선순위

우선순위	연산자	설명
1	(), [], {}	튜플, 리스트, 딕셔너리, 집합 생성
2	A[], A[:], A(인수...), A.속성	리스트(튜플) 인덱싱, 슬라이싱, 함수 호출, 속성 참조
3	await A	await 표현식
4	**	거듭제곱
5	+A, -A, ~A	양의 부호, 음의 부호, 비트 NOT
6	*, /, //, %	곱셈, 나눗셈, 몫, 나머지
7	+, -	덧셈, 뺄셈
8	<<, >>	비트 시프트
9	&	AND
10	^, |	XOR, OR
11	In, not in, is, is not, 비교 연산자	구성원 연산자, 식별 연산자, 비교 연산자(<, <=, >, >=, !=, ==)
12	not, or, and	논리 NOT, 논리 OR, 논리 AND
13	if else	조건부 표현식
14	lambda	람다 표현식

에러처리
Exception

Except

```
try:
    수행문장
except 에러 as e:
    에러처리
```

Error	설명
SyntaxError	잘못된 문법
NameError	참조변수 없음
ZeroDivisionError	0으로 나눌 수 없음
IndexError	인덱스 범위 벗어남
ValueError	참조값 없음
KeyError	키 없음(딕셔너리)
AttributeError	모듈, 클래스의 잘못된 속성 사용
FileNotFoundError	파일 못 찾음
TypeError	타입 안 맞음

에러처리
Exception

Raise

의도적으로 에러를 만드는 방법

Assert

조건이 참이 아니라면 에러 발생

모듈
module

import

모듈이란 변수나 함수 또는 클래스를 모아 놓은 파일입니다.

import 모듈

from 모듈 import 요소

* (all)

as

as → 별칭

import tensorflow as tf

import numpy as np

import pandas as pd

파일 입출력
File input/output

txt파일

텍스트 파일 데이터 입출력

명령어	설명	비고
r	읽기	
w	쓰기	파일 내용 지움
a	쓰기	파일 끝에 추가
x	쓰기	파일이 있으면 에러
w+	읽기/쓰기	파일 내용 버림
r+	읽기/쓰기	파일 내용 유지
a+	읽기/쓰기	파일 내용 유지
x+	읽기/쓰기	파일 있으면 에러

파일 입출력
File input/output

txt파일

텍스트 파일 데이터 입출력

```
f = open(파일경로, 'r', encoding='utf-8')

# 한 줄 읽기
while True:
    data = f.readline()
    if not data:
        break
    print(data)
f.close()
```

```
f = open(파일경로, 'r', encoding='utf-8')

# 여러 줄 읽기
data = f.readlines()
for line in data:
    print(data)
f.close()
```

파일 입출력
File input/output

txt파일

텍스트 파일 데이터 입출력

```
f = open(파일경로, 'w', encoding='utf-8')

# 한 줄 쓰기
f.write(작성글)
f.close()
```

```
f = open(파일경로, 'w', encoding='utf-8')

# 여러 줄 쓰기
data = f.readlines()
f.writelines(iterable객체)
f.close()

ex) list_data = ['Hello, Python₩n', '안녕', '파이썬₩n']
```

파일 입출력
File input/output

with

해당 들여쓰시(intent)가 끝나는 지점에서 자동으로 close 역할을 해주는 문법

```
with open(파일경로, 'r+', encoding='utf-8') as f:
    f.write('데이터 확인')
    print(f.readlines())
```

파일 입출력
File input/output

csv

CSV파일이란 ,(쉼표)로 액셀의 셀들을 구별하는 파일 형식입니다.

```
import csv

with open(파일경로, 'r', encoding='utf-8') as f:
    read_data = csv.reader(f)
    for data in read_data:
        print(data)
```

파일 입출력
File input/output

json

JSON파일이란, {속성: 값}으로 이루어져 있는 모습이며, 우리가 이전에 배운 사전형 자료형(dictionary)와 같은 형태입니다.

```
import json

with open(파일경로, 'r', encoding='utf-8') as f:
    data = json.load(f)
    print(type(data), data)
```

```
import json

data = {'response': 'json', 'code': 200, 'result': {'data': '읽기 성공'}}
json_data = json.dumps(data, ensure_asci=False)
print(type(json_data), json_data)
```

유니코드로 저장되는 것을 막음

함수
def

def

```
def 함수이름(parameter):
    수행문장
    return 값
```

Parameter 매개변수라고도 함

Parameter / Argument

매개변수: 미지수의 변수 → 함수에서 받는 값

인자: 정해진 값 → 함수 호출 시 넣어주는 값

함수
def

args

args: *

kwargs

kwargs: **

함수
def

람다식

람다 표현식은 함수를 간단하게 표현하는 문법입니다.

lambda 매개변수: 식

(lambda 매개변수: 식)(인자값)

함수
def

재귀함수

정의된 함수가 자신의 함수를 다시 호출하는 방식입니다.

```
def f():
    return f()
```

재귀함수를 구현할 때는 주의해야할 점들이 많습니다.

무엇보다 자신을 지속적으로 호출할 경우 허용하는 범위를 초과하면 자동으로 멈추게 됩니다.

파이썬 기본설정에서는 1000번을 초과할 수 없게 되어 있습니다.

함수
def

1. 팩토리얼(factorial) 기능을 만들어 주세요.