파이썬은 인터프리터 언어이다

코드를 한줄 한줄 실행함

파이썬은 인간다운 언어이다

파이썬은 문법이 쉬워 빠르게 배울 수 있다

파이썬은 무료이지만 강력하다

파이썬은 간결하다

파이썬은 프로그래밍을 즐기게 해준다

파이썬은 개발 속도가 빠르다

파이썬 숫자형

정수, 실수 생략

8진수 0o 또는 0O

숫자0, 영어 o로 시작하면 된다

a = 0o177

16진수는 0x로 시작하면 된다

a = 0x1ff

연산자

+, -, \*, /, % 생략

제곱 연산자 \*\*

2 \*\* 3 = 8

나눗셈 후 몫을 반환하는 //연산자

7 / 4 = 1.75

7 // 4 = 1

=============================문자열=============================

파이썬 문자열

''

""

""" """

''' '''

4가지 방법이 있다

' "" '

" '' "

' \' ' > 백슬래시를 사용하여 따옴표를 문자열에 포함시킨다

\n으로 줄바꿈

""" or '''를 사용시 엔터를 사용하여 입력해도 그대로 들어감

multiline="""

... Life is too short

... You need python

... """

print(multiline)

Life is too short

You need python

이스케이프 코드 정리

\n | \t | \\ | \' | \"

줄바꿈, 탭, 역슬래시, 작은따옴표, 큰따옴표

문자열 연산

합 | a+b = ab

곱 | a\*2 = aa

길이 len(a) = 1

a[0] = a >> 문자열을 리스트처럼 뽑아 쓸 수 있음

a = "String is List?"

a[-2] = 's'

a[-4] = 'L'

마이너스를 써도 뒤에서부터 문자를 가리킨다

문자열 슬라이싱

a[0:6] = 'String' // 0부터 6이전까지 문자를 뽑아내겠다는 뜻

a[7:8] = 'is'

a[0:] = 'String is List?' // 끝부분 생략하면 시작부터 끝까지 뽑아냄

a[:6] = 'String' // 앞부분 생략하면 끝 번호까지 뽑아냄

a[:] = 'String is List?' // 시작, 끝 생략하면 문자열 전부 출력

a[10:-2] = 'Li' // 여기서도 마이너스 기호 사용 가능

문자열의 요소 변경은 불가능

그래서 문자열을 슬라이스 하여 중간에 요소를 추가하는 방법을 쓴다

문자열 포매팅

문자열에 특정 값을 넣고싶을때 문자열 포매팅을 쓴다

>>> "I wish number %d programmer" % 1

'I wish number 1 programmer

>>> "I wish number %s programmer" % "one"

'I wish number one programmer'

숫자는 %d, 문자는 %s를 쓴다

"I ate %d apple and ate %d pineapple" % (1,2)

2개 이상 넣으려면 괄호와 콤마를 사용하면 된다

문자열 포맷 코드

%s 문자열(String)

%c 문자 1개(character)

%d 정수(Integer)

%f 부동소수(floating-point)

%o 8진수

%x 16진수

%% Literal % (문자 % 자체)

%s에 숫자를 넣어도 자동으로 문자열로 변환해준다

%d 와 %를 같이 쓸 땐 %d%%를 쓴다

포맷 코드와 숫자 함께 사용하기

>>> "%10s" % "ho"

' ho'

10개인 문자열에서 들어가는 값을 오른쪽으로 정렬하고, 나머지는 공백 전환

>>> "%-10smeo" % 'ho'

'ho meo'

왼쪽 정렬은 마이너스를 쓴다

>>> "%10.4f" % 3.42134234

' 3.4213'

소수점 정렬은 4번쨰 자리까지 표시, 10개 문자열에서 오른쪽정렬 까지 하는 모습이다

format 함수를 사용한 포매팅

>>> "I wish number {0} programmer".format(1)

'I wish number 1 programmer

문자열 뒤에 format함수의 인덱스 사용하여 숫자를 넣음

>>> "I wish number {0} programmer {1}".format(1, "is good")

'I wish number 1 programmer is good

여러개 넣을때는 인덱스의 숫자가 점점 올라간다

>>> "I wish number {number} programmer {good}".format(number=3, good=3)

'I wish number 3 programmer 3

이름을 사용하여 넣는 방법도 있다

>>> "I wish number {number} programmer {0}".format(number=3, 3)

'I wish number 3 programmer 3

이름과, 인덱스를 혼용하여 넣는 방법도 있다

"{0:>10}".format("hi") 오른쪽 정렬

' hi'

"{0:<10}".format("hi") 왼쪽 정렬

'hi '

"{0:^10}".format("hi") 가운데 정렬

' hi '

소수점 표현

>>> y = 3.42134234

>>> "{0:0.4f}".format(y) 소숫점 4자리수까지

'3.4213'

>>> "{0:10.4f}".format(y) 10공백 오른쪽 정렬

' 3.4213'

>>> "{{ and }}".format() format에서 중괄호 사용법은 {{ }} 2번씩 쓰는것

'{ and }'

f문자열 포매팅

>>> name = '홍길동'

>>> age = 30

>>> f'나의 이름은 {name}입니다. 나이는 {age}입니다.'

'나의 이름은 홍길동입니다. 나이는 30입니다.'

괄호에 +,- 와 같은 수식이 사용이 가능하다

딕셔너리는 f 문자열 포매팅에서 다음과 같이 사용할 수 있다.

>>> d = {'name':'홍길동', 'age':30}

>>> f'나의 이름은 {d["name"]}입니다. 나이는 {d["age"]}입니다.'

'나의 이름은 홍길동입니다. 나이는 30입니다.'

정렬도 비슷하나 뭐 format이나 다름없다

>>> f'{"hi":<10}' # 왼쪽 정렬

'hi '

>>> f'{"hi":>10}' # 오른쪽 정렬

' hi'

>>> f'{"hi":^10}' # 가운데 정렬

' hi '

공백 채우기는 다음과 같이 할 수 있다.

>>> f'{"hi":=^10}' # 가운데 정렬하고 '=' 문자로 공백 채우기

'====hi===='

>>> f'{"hi":!<10}' # 왼쪽 정렬하고 '!' 문자로 공백 채우기

'hi!!!!!!!!'

소수점은 다음과 같이 표현할 수 있다.

>>> y = 3.42134234

>>> f'{y:0.4f}' # 소수점 4자리까지만 표현

'3.4213'

>>> f'{y:10.4f}' # 소수점 4자리까지 표현하고 총 자리수를 10으로 맞춤

' 3.4213'

문자열 관련 함수들

count

a = "hobby"

a.count('b')

2

문자열중 b의 개수를 돌려준다

a.find('b')

2

문자열중 b가 처음 나온 위치를 반환한다

만약 없다면 -1을 반환한다

a.index('b')

find처럼 b가 처음 나온 위치를 반환한다

find와 다른 점은 index는 없으면 오류를 발생시킨다

","join(a)

'h,o,b,b,y'

문자열 각각 사이에 ','를 삽입한다

>>> ",".join(['a', 'b', 'c', 'd'])

'a,b,c,d'

리스트나 튜플도 입력으로 사용할 수 있다

대문자 a.upper() // 'HOBBY'

소문자 a.lower() // 'hobby'

공백 지우기

a=" hi "

a.rstrip() // ' hi' 오른쪽 공백 지움

a.lstrip() // 'hi ' 왼쪽 공백 지움

a.strip() // 'hi' 양쪽 공백 지움

문자열 바꾸기

a = "hobby is nope"

a.replace("hobby", "life")

'life is nope'

문자열 나누기

a.split()

['hobby', 'is', 'nope']

b="a:b:c:d"

b.split(':')

['a', 'b', 'c', 'd')

=============================문자열=============================

=============================리스트=============================

a = [1, 2, 3, 4, 5]

a[0] // 1

a[2] // 3

a[-1] // 5

a = [1,2,3,['a','b','c']]

a[0] // 1

a[-1][0] // 'a'

a[-1][1] // 'b'

a = [1, 2, 3, 4, 5]

a[0:2] // [1,2] 슬라이싱도 가능하다

a[:2] // [1,2]

a[2:] // [3,4,5]

중첩된 리스트에서도 슬라이싱이 가능하다

리스트 더하면 문자열 처럼 리스트가 붙는다

리스트 \* 는 문자열처럼 곱한 만큼 반복되어 새로운 리스트 생성

리스트의 길이 len(list)

리스트는 요솟값을 바꿀 수 있다

a=[1,2,3]

a[2] = 4 // a=[1,2,4]

del로 리스트 요소 삭제

del a[1] // [1,4]

del a[1:] // [1] 슬라이싱을 사용하여 여러개 삭제도 가능하다

a.append(4) 리스트 뒤에 요소 추가

a.sort() 리스트 요소 순서대로 정렬, 문자열도 알파벳 순서

a.reverse() 리스트를 역순으로 뒤집음, 정렬후 역순이 아닌 그냥 뒤집는것

a.index(x) 리스트 x에 값이 있으면 x의 위치값을 돌려줌, 없으면 오류

a.insert(0,x) 0번방에 x이라는 값을 넣는다, 넣은 다음 뒤의 값들은 뒤로 밀린다

a.remove(x) 리스트에서 첫 번째로 나오는 x를 삭제하는 함수

a.pop() 리스트의 맨 마지막 요소를 돌려주고, 삭제한다 / pop에 x를 넣으면 x번째 요소를 돌려주고, 삭제한다

그러니까 pop은 방 번호로 삭제, remove는 요소의 값을 찾아서 삭제

a.count(x) 리스트에서 x의 개수를 돌려줌

a.extend([a,b]) a리스트에서 [a,b]를 더한다 // 그냥 리스트 더하기랑 같다...

=============================리스트=============================

============================= 튜 플 =============================

리스트는 [ ] // 값을 바꿀 수 있다

튜플은 ( ) // 값을 바꿀 수 없다

t1 = ( )

t1 = (1,)

t2 = 1,2,3

1개 요소가 있을땐 마지막에 콤마 필요

괄호를 생략해도 무방하다

튜플은 거의 사용 안함

위에서 보았던 t[0](인덱싱)

t[1:](슬라이싱)

t1+t2 (더하기)

t1\*3(곱하기)

len(t1) 길이

값을 변경하지 않는 이런 것들은 사용 가능하다

============================= 튜 플 =============================

============================ 딕셔너리 ============================

딕셔너리는 key와 value를 가진다

a = {1:'a'}

여기서 1은 key 'a'는 value이다

a = {1:[a,b,c]} 딕셔너리 안에 리스트도 가능하다

딕셔너리 쌍 추가하기

>>> a = {1: 'a'}

>>> a[2] = 'b'

{1: 'a', 2: 'b'}

{1: 'a'} 딕셔너리에 a[2] = 'b'와 같이 입력하면 딕셔너리 a에 Key와 Value가 각각 2와 'b'인 2 : 'b'라는 딕셔너리 쌍이 추가된다.

>>> a['name'] = 'pey'

{1: 'a', 2: 'b', 'name': 'pey'}

딕셔너리 a에 'name': 'pey'라는 쌍이 추가되었다.

>>> a[3] = [1,2,3]

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

딕셔러니 요소 삭제하기

del a[key]

del a[1]

{2: 'b', 'name': 'pey', 3: [1, 2, 3]}

key를 사용해 value 얻기

a[2] / b

a['name] / 'pey'

딕셔너리 주의사항

중복되는 key값을 설정하면 하나를 제외한 나머지는 모두 무시

key에는 list를 쓸 수 없다, 쓰면 오류남

딕셔너리 관련 함수들

a.keys()는 딕셔너리 a의 key만 모아서 dick\_keys 객체를 돌려준다

a = {'name': 'pey', 'phone': '0119993323', 'birth': '1118'}

a.keys()

dict\_keys(['name', 'phone', 'birth'])

dict\_keys 객체를 리스트로 변환하려면 list(a.keys())를 사용하면 된다

list(a.keys())

a.values()는 딕셔너리 a의 value만 모아서 dict\_values를 반환한다

a.items()는 딕셔너리의 key와 value의 쌍을 튜플로 묶은 값을 dict\_items 객체로 돌려준다

a.clear() / key: Value 쌍 모두 지우기

key로 value 얻기(get)

a.get('name') / 'pey'

만약 get에서 키값이 없는거라면 None을 반환한다

a.get(x, default) 를 사용하면 x가 없을 시 디폴트 값을 반환한다

해당 key가 딕셔너리에 있는지 조사하기 in

'name' in a / True

'aa' in a / False

============================ 딕셔너리 ============================

=========================== 집합자료형 ===========================

=========================== 집합자료형 ===========================

============================= if문 =============================

and, or, not

if a or b:

x or y x와 y 둘중에 하나만 참이어도 참이다

x and y x와 y 모두 참이어야 참이다

not x x가 거짓이면 참이다

x in s, x not in s

in not in

x in 리스트 x not in 리스트

x in 튜플 x not in 튜플

x in 문자열 x not in 문자열

============================= if문 =============================

============================ while문 ============================

while 정수>10:

while 리스트:

continue

break

무한루프

============================ while문 ============================

============================= for문 =============================

for 변수 in 리스트(또는 튜플, 문자열):

리스트 넣을 시 변수에는 리스트의 요소가 들어간다

for i in range(5) 5까지

for i in range(1,5) 1부터 5까지

for i in range(5,0, -1) 5부터 0까지 -1씩 까짐

리스트 내포 사용하기

이건 좀 기니까 예시로 대체

numbers = [1, 2, 3, 4, 5]

result = []

for n in numbers:

if n % 2 == 1:

result.append(n\*2)

for문을 >>> result = [n\*2 for n in numbers if n%2 == 1] >>> 로 대체한다

============================= for문 =============================

============================ 파일RWA ============================

f = open('test.txt','w')

f.write('파일 쓰기')

f = open('test.txt','r')

f.readline() / 한 줄 읽기

f.readlines() / 한 줄씩, 리스트 형태로 읽기

f.read() / 파일 내용 전체를 문자열로

f = open('test.txt','a')

f.write('기존 파일 그대로, 이어서 쓰기')

f.close() / 사용하면 닫아줘야 다음 파일 사용시 에러가 없다

with문을 사용하면 자동으로 해줌

with open('test..txt', 'w') as f:

f.write('텍스트')

============================ 파일RWA ============================

============================= 클래스 =============================

class cal():

def \_\_init\_\_(self): # 생성자

self.a = a

sefl.b = b

def add(): # 메소드

return self.a + self.b

class copycal(cal): # 클래스 상속

pass

class copycal(cal): # 메소드 오버라이딩 method overriding

def add():

return self.a - self.b

a = cal(1,2)

print(a.add())

b = copycal(2,1)

print(b.add()) # 1

참고로 메소드 오버로딩은 똑같은 함수를 불러오지만

오버라이딩 처럼 재정의가 아닌 매개변수의 유형과 개수를 다르게 하여 사용하는것이다

그러니까 클래스 내의 같은 이름의 함수에서 매개변수만 다르게 받는 것

============================= 클래스 =============================

============================== 모듈 ==============================

mod.py 가 같은 디렉토리에 있다면

import mod 를 사용하여 불러올 수 있다.

그냥 import 는 a.mod.func() 이렇게 써야함

또는 모듈의 함수만 가져오고 싶을 땐

from mod import add

from을 써서 필요한 함수만 가져 올 수 있다.

from mod import \* 로 모든 함수를 가져 올 수 있다.

from을 쓰면 a.func()로 바로 쓸 수 있음

그러나 모듈을 불러오게 된다면

클래스의 함수가 아닌 밖의 print와 같은 문장은 바로 실행하게 되는데

if \_\_name\_\_ == "\_\_main\_\_": 를 사용하여 이 아래에 둠으로써 다른 파일에서 실행 시 바로 시작하지 않도록 한다

sys 모듈을 사용하여 지정한 디렉터리가 사용이 가능하다

import sys

sys.path.append("C:/")

이러면 C 안에있는 파이썬 파일이 사용 가능하다

또는 PYTHONPATH 환경변수를 사용하는 방법

cmd 명령어 창에서

set PYTHONPATH=C:\

이러면 C 안에있는 파이썬 파일이 사용 가능하다

============================== 모듈 ==============================

============================== 패키지 ==============================

패키지는 토트 (.)를 사용하여 파이썬의 모듈을 계층적(디렉터리 구조)으로 관리할 수 있게 해준다

디렉토리가 이렇다고 가정하자

텍스트이(가) 표시된 사진

자동 생성된 설명

set PYTHONPATH=C:/doit 으로 지정

import game.sound.echo // 모듈을 import 하는 방법

game.sound.echo.echo\_test()

from game.sound import echo // sound까지 들어가서 echo모듈을 불러오는 것

echo.echo\_test

from game.sound.echo import echo\_test // echo 까지 들어가서 echo\_test 함수를 불러옴

echo\_test()

단 import로 echo\_test 함수까지 불러올 수 없다. 오로지 모듈 또는 패키지여야만 한다

\_\_init\_\_.py 파일은 해당 디렉터리가 패키지의 일부임을 알려주는 역할을 한다.

만약 위의 폴더에서도 \_\_init\_\_.py 파일이 없으면 패키지로 인식하지 않는다.

3.3버전부터는 없어도 패키지로 인식 하나 하위버전 호환을 위해 파일 생성하는게 안전하다고 한다

from game.sound import \* 하면 에러가 난다

\_\_init\_\_.py 파일에 \_\_all\_\_ 변수를 설정하고, import 할 수 있는 모듈을 정의해 주어야 한다

예시로는 \_\_init\_\_.py에 \_\_all\_\_ = [‘echo’] 를 넣으면 된다…

from game.sound.game import \* 은 에러가 안남

이미 echo 모듈까지 갔기 때문에 echo모듈의 함수들을 불러오기 때문…

만약 다른 디렉토리(ex:graphic)에서 sound의 echo.py를 사용하고 싶을 땐

위와 동일하다

graphic의 render.py에

from game.sound.echo import echo\_test 또는

from ..sound.echo import echo\_test

여기서 (..)은 부모디렉터리를 의마한다

graphic과 sound는 같은 디렉토리를 사용하므로 (..)를 사용하여 위와 같은 import가 가능하다

(.) 은 현재 디렉터리, (..)은 부모 디렉터리를 의미한다

============================== 패키지 ==============================

============================= 예외처리 =============================

기본적으로

try:

except:

로 사용한다

except를 여러개 사용이 가능하다

단 발생 오류 내용이 달라야 한다

except Exception

except Exception as e

오류 문구는 이렇게 사용이 가능하다

except에 pass를 사용하면 넘어가기 가능

rasie NotImplementedError 을 사용하여 오류를 강제로 발생시킬 수 있다  
============================= 예외처리 =============================

============================= 내장함수 =============================

파이썬엔은 유용한 내장 함수가 많다

그걸 잘 활용하면 좋을듯…

abs 절대값

all 요소들이 모두 참이면 True 아니면 False // and같은거

any 요소들이 하나라도 참이면 True 아니면 False // or같은거

chr(i) 아스키 코드값을 입력받아 그 코드에 해당하는 문자 출력

dir 자체적으로 가지고 있는 변수나 함수를 보여줌

============================= 내장함수 =============================

============================= 라이브러리 =============================

============================= 라이브러리 =============================