\*\*: 제곱

//: 몫의 정수값

문자열

-따옴표 사용법:

“Python’s food”

‘”wow!” he says.’

“\”Python’s food\”” (\를 따옴표 옆에 사용)

줄 바꿈:

* “Life is too short\n You need Python”
* “”” 사용
* ‘’’ 사용

문자열 덧셈, 문자열 곱셈

문자열 길이 구하기: len()

문자열 인덱싱: a[0] … a[n-1] or a[-1] … a[-n]

문자열 슬라이싱:

* a[0:4] (수식: 0 <= a < 4)
* a[:4]
* a[5:]

문자열 포매팅

* “I have %s dogs” % 숫자, 문자열, 변수
* “I have %d dogs and %s cats” % (5, “three”)

Format 함수

* “I have {0} dogs and {1} cats.” .format(num1, num2)

문자열 관련 함수

Count (문자 개수 세기)

* a.count(‘b’)

find (위치 인덱스 알려주기, 단 없는 것 찾으면 -1 반환)

* a.find(‘b’)

index(위치 인덱스 알려주기, 단 없는 것 찾으면 오류)

* a.index(‘b’)

join(문자열 삽입)

* “,”.join(‘abcd’) -> a,b,c,d

upper(소문자->대문자)

* a.upper()

lower(대문자->소문자)

* a.lower()

capitalize(첫 번째 알파벳 capitalize)

* a.capitalize()

endswith(문자열이 “문자열”로 끝이 나는지 확인)

* a.endswith(“wow”) / it’s wow면 true
  + a.endswith((“wow”, “crazy”)) / [괄호 주의] it’s wow 또는 it’s crazy이면 true

startswith(문자열이 “문자열”로 시작하는지 확인)

* a.startswith(“wow”) / wow it’s면 true
  + a.startswith((“wow”, “crazy”)) / [괄호 주의] wow it’s 또는 crazy it’s이면 true

lstrip, rstrip, strip (왼쪽, 오른쪽, 양쪽 공백 지우기)

* a.rstrip() ‘ hi ‘ -> ‘hi ‘

replace (문자열 바꾸기)

* a.replace(“Life”, “Your leg”)

split(문자열 나누기, default: 스페이스바)

* a.split() -> [Life is too short] -> [‘Life’, ‘is’, ‘too’, ‘short’]
* b.split(‘:’) -> [a:b:c:d] -> [‘a’, ‘b’, ‘c’, ‘d’]

len(문자열, 리스트, 튜플 길이)

* len(a) -> 3

str(문자열로 바꾸기)

* str(a) -> 3 -> ‘3’

리스트

생성

A = list() (비어있는 리스트 생성)

A = [] (비어있는 리스트 생성)

인덱싱

A = [a, b, [1, 2, 3]]

* A[0], A[1]… or A[-1]…
* A[2][0]

슬라이싱

A = [a, b, [1, 2, 3]]

* A[:2] -> [a, b]
* A[2][:2] -> [1, 2]

리스트 연산

* +: [1, 2, 3] + [4, 5, 6] = [1,2,3,4,5,6]
* \*: [1, 2, 3] \* 3 = [1,2,3,1,2,3,1,2,3]

리스트 길이 (a = [1, 2, 3])

* len(a) = 3

리스트 수정과 삭제(a = [1, 2, 3])

* a[2] = 4 -> [1, 2, 4]
* del a[1] -> [1, 3]

리스트 관련 함수 (a = [1, 2, 3])

Append(요소 추가)

* a.append(4) -> [1, 2, 3, 4]

sort(정렬)

* a**.sort**() -> [1, 3, 2] -> [1, 2, 3]
  + 반환식: data = **sorted** (a)

reverse(뒤집기)

* a.reverse() -> [3, 2, 1]

index(위치 반환)

* a.index(2) -> 1

insert(삽입)

* a.insert(0, 4) -> [4, 1, 2, 3]

remove(첫 번째로 나오는 x 제거)

* a.remove(3) -> [1, 2]

pop(반환 후 삭제)

* a.pop() -> [1, 2] (맨 마지막 요소)
* a.pop(1) -> [1, 3] (1번째 요소)

count(요소 개수 세기)

* a.count(1) -> 1

extend(리스트 확장)

* a.extend([4, 5]) -> [1, 2, 3, 4, 5]

min(최솟값)

* min(list) / list에서 최솟값 반환

max(최댓값)

* min(list) / list에서 최댓값 반환

sum(합)

* sum(list) / list의 합을 반환

튜플

특징:

1. [ ]를 사용하는 리스트와는 달리 ( )를 사용 (사용하지 않아도 무방)
2. 리스트와는 달리 안에 값 생성, 수정, 삭제 불가

생성

* T1= ( )
* T1 = tuple(…) (…를 튜플로 만듦)
* T2 = (1, ) (하나 생성 시 콤마 붙여야 함)
* T3 = (1, 2, 3) or 1, 2 ,3

튜플 함수

* 리스트와 동일 (추가&삭제 함수 제외)

패킹/언패킹

* 패킹: 일반적인 리스트 & 튜플 생성 방식
  + Num = 1,2,3,4,5
  + Num = [1,2,3,4,5]
* 언패킹: list 혹은 tuple로부터 len(list or tuple)만큼을 풀어서 각 변수에 대입
  + Num = 1,2,3
  + A,B,C = Num -> (A=1, B=2, C=3)
  + A,\_,\_ = Num -> (A=1, B=undefined, C=undefined)
  + \*a= num -> (a=[1,2,3]) //리스트 형태로 나머지들 반환

딕셔너리

Dic = {key1:value1, key2:value2, key3:value3, …} (key: 변하지 않음, value: 변하거나 변하지 않음)

특징

* key:value
* 리스트, 튜플과 같은 인덱싱 슬라이싱 기법 사용불가
* Key에 리스트 사용불가, 튜플 사용가능

생성

* A = dict()

추가

* **A[‘name’] = ‘pey’** -> [‘name’:’pey’] A[2] = ‘010’ -> [‘name’:’pey’, 2:’010’]

삭제

* **del A[2]** -> [‘name’:’pey’]

딕셔너리 관련 함수

a.keys() (key 리스트 생성)

- dict\_keys([……]) 생성됨 (반환 x)

list(a.keys())

* 반환(o)

a.values() (value 리스트 생성)

- dict\_values([……]) 생성됨 (반환 x)

a.items() (key, value쌍을 튜플로 반환)

- dict\_items([(‘name’, ‘pey’), (‘phone’, ‘…’), (‘birth’, ‘…’)])

a.clear() (key:value 쌍 모두 지우기)

a.get(key) (key로 value 얻기)

- a[‘key’]와 같음 (단지, 없는 값 반환 시 오류와 None 반환 차이)

- a(‘foo’, ‘bar’) (foo가 없고 bar가 있으면 bar 반환)

Key in a (key가 딕셔너리에 있는지 조사)

* ‘name’ in a -> True or False

집합 자료형(set)

특징

* 집합을 쉽게 처리하기 위한 자료형
* 중복을 허용하지 않는다
* 순서가 없다 (인덱스로 값 얻기 불가능)

생성

* S = set()
* S = set(“Hello”)
* S = set([1, 2, 3])

Set 관련 함수 (교집합, 합집합, 차집합)

교집합

* S1 & S2
* S1.intersection(S2)

합집합

* S1 | S2
* S1.union(S2)

차집합

* S1 – S2 or S2 – S1
* S1.difference(S2)

S1.add (값 추가)

* S1.add(4) -> ([1,2,3]) -> ([1,2,3,4])

S1.update (값 여러 개 추가)

* S1.update([4, 5, 6]) -> ([1, 2, 3, 4, 5, 6])

S1.remove (특정 값 제거)

* S1.remove(2) -> ([1, 3])

불 자료형

특징

* True, False 두 값만을 가질 수 있음
* 비어 있지 않음, 1 = True, 비어 있음, 0, None = False

Bool 연산

* bool(‘python’) -> True

파이썬의 변수

파이썬의 변수는 복사하면 두 값의 주소 자체가 같다.

* A = B를 할 경우 A와 B의 주소는 같아진다. (즉, A값을 바꾸면 B값도 바뀐다)

주소가 다른 변수를 만들기

* 리스트의 [:] 이용
  + A = [1, 2, 3], B = A[:] -> 주소가 다른 변수 생성
* Copy 모듈 이용
  + From copy import copy 모듈 이용 후: a = [1, 2, 3], b = copy(a)

여러 변수 생성

* A, B = (‘python’, ‘life’)
* [A, B] = [‘python’, ‘life’]

SWAP

* A, B = B, A

If문 (조건문)

If ~ elif ~ else ~ (들여 쓰기)

비교연산자 (> < == != >= <=)

And, or, not

* If money or card
* X or y
* x in, x not in

조건부 표현식: 조건문이 참인 경우 if 조건문 else 조건문이 거짓인 경우

* message = “success” if score >= 60 else “failure”

While문

사용: while 조건문

강제로 빠져 나가기:

* break

while문으로 다시 돌아가기:

* continue

무한 루프:

* while True:

For문

사용: for 변수 in (문자열, 리스트, 튜플 등)

* for i in a: (i에 a의 요소가 대입되어 계속 수행)
* for (first, last) in a: (first와 last 튜플에 (a,b)의 a와 b가 대입되어 계속 수행)

range 함수:

* range(stop) / range(start, stop, step)
* range(10) == range(0, 10)
* range(0,99,2) == 0부터 99까지 2씩
  + 0부터 9까지의 범위
* Ex) for i in range(1, 11) print(i) -> 1, 2,…
* Ex) for I in range(len(a))

리스트 내포: (num = [1, 2, 3, 4])

* Result = [num \* 3 for I in a]
* Result = [num \* 3 for I in a if num > 5] (조건문 추가)
* Result = [x \* y for x in range(2, 10) for y in range(1, 10)]

함수

* Def 함수이름 (매개변수):

매개변수 지정:

* Def 함수이름 (b = 5, a = 4) == def 함수이름 (a = 4, b = 5)

인수 개수 무작위일 시:

* Def 함수이름 (\*args) (인수의 개수에 따름)
  + Ex) for i in args
* Def 함수이름 (choice, \*args) (첫 번째 인수, 그 후의 인수들)\

키워드 매개변수:

* Def 함수이름 (\*\*매개변수)
* Ex) def print\_name (\*\*kwargs)
  + Print\_name (name = ‘ben’)
    - {‘name’ : ‘ben’}
* 키워드 매개변수에 보낼 시 딕셔너리로 만들어 줌

함수의 결과값은 하나:

* Add\_and\_mul: Return a+b, a \* b

>>> (a + b, a \* b)인 튜플

* Result1, result2 = add\_and\_mul(a+b, a\*b)
  + Result1과 result2 각각에 값 줌

매개변수 초기값(기본값) 설정:

* Def function (a, b, man = True):
  + 인수: function(5, 4)
  + 인수: function(5, 4, True or False)
    - 주의: 초기값 설정할 매개변수는 맨 끝에 두기

함수 안에서 함수 밖 변수 변경하기:

* Return으로 값 받기
* Global a 사용하기
  + Global 함수 사용 시 전역 변수의 a를 사용하게 됨

함수의 특징:

* 키워드값
  + parameter와 argument의 키워드가 같으면 서로 순서가 달라도 알맞게 적용됨

Lambda:

* Def와 같은 “함수 생성” 함수, 그러나 간략한 함수를 만들 때 사용
* Lambda 매개변수1, 매개변수2: 표현식
* Ex) mul = lambda a, b: a \* b

Result = mul(a, b) 🡪 mul(9,4) == 36

사용자 입력과 출력

Input()

* Ex) a = input(“입력하시오: ”)
  + Input()은 모든 것을 문자열로 받아들임

Print()

* Print(“life” “is” “too”)
  + Lifeistoo
* Print(“life”,”is”,”too”)
  + Life is too

한 줄에 입력하기:

For I in range(10):

Print(I, end = ‘ ‘)

파일 읽고 쓰기

**파일 생성하기:** **open**

F = open(“new.txt, ‘포맷’) [포맷: r, w, a] r = 읽기, w = 쓰기, a = 추가

**파일 닫기:** **close**

f.close() (생략해도 되지만 쓰면 오류 방지)

**파일 쓰기:** **write**

f.write(data)

**파일 첫번째 줄 읽기: readline** (한 줄 읽고 다음 줄을 타겟으로 변경)

Line = f.readline()

**파일 모든 줄 읽기:** **readlines** (모든 줄을 읽고 한 줄 씩 나눠진 리스트로 반환)

Lines = f.readlines()

* Lines는 [“1번째 줄\n”, “2번째 줄\n”, ……]

**줄바꿈(\n) 없애기: strip**

Line = line.strip()

**파일 전체 읽기: read** (모든 줄을 문자열로 읽음)

Line = f.read()

**파일에 내용 추가하기: ‘a’**

f = open(“new.txt”, ‘a’)

f.write(data)

**with문: with ~ as ~** (with문 벗어나면 f.close가 자동으로 사용됨)

with open(“new.txt”, ‘w’) as f: ~~~

클래스

클래스는 마치 제품을 만들어내는 3D프린터와 같다

클래스 생성:

* Class cookie:

(method 함수들)~

* + 여기서 cookie는 클래스라고 불린다

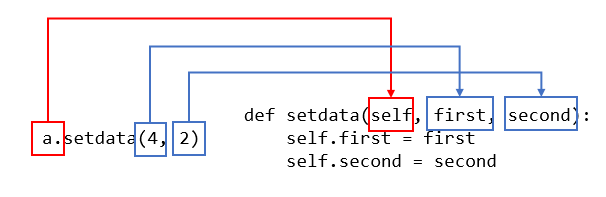
객체 생성:

* A = cookie()
  + 여기서 A는 cookie의 인스턴스라고 불린다

메서드 호출:

* 도트 함수를 써야 함: a.setdata()처럼 “.”을 써야 함
* 클래스로 메서드 호출: foucal.setdata(a, …)
  + 객체를 매개변수에 써줘야 함

메서드의 첫 매개변수로 사용되는 self:

* 
  + 객체 자신이 첫 매개변수로 전달됨

상속:

* class a(self, c, d):

class b(a):

a.\_\_init\_\_(self, c, d)

- b가 a의 기능을 상속 받음

다중 상속:

* class a:

def \_\_init\_\_(self, c, d)

class b:

def \_\_init\_\_(self, e, f)

* class c(a, b)

피클

* 피클은 text file로 저장하는 것이 아닌, list, tuple, dictionary의 형태로 binary 형식으로 저장하는 것
* 사용법:
  + import pickle (피클 모듈 불러오기)
  + 쓰기: (list, tuple, dictionary를 ltd라고 가정)
    - pickle\_file = **open(“**p\_file.pickle**”, “wb”)**

**pickle.dump(**ltd, pickle\_file**)**

* + 읽기:
    - pickle\_file = **open(“**p\_file.pickle**”,** **“rb”)**
    - variable = **pickle.load(**pickle\_file**)**

예외처리

* 에러가 발생했을 때 명시할 에러를 정하는 것
* try:

1. except 에러명:
   * + 해당 에러에 대해서 ~라고 처리
2. except 에러명 as 변수:
   * + ex) except as err
     + 에러명에 해당하는 에러의 코멘트를 변수에 넣음
3. except:
   * + 나머지에 대해서는 ~라고 처리
4. except **Exception** as 변수:
   * + Exception에 해당하는 에러명의 코멘트를 변수에 넣음

* 예외상황(에러) 발생시키기
* **raise 에러명**
  + 에러명에 해당하는 에러를 발생시킨다
  + **except 에러명**을 통해서 에러 처리를 해줘야 한다
* 사용자 정의 예외처리의 경우 class로 에러를 만들어 줘야 한다
* **finally:**
  + 에러의 발생 여부와 관계 없이 마지막에 원하는 코드를 실행시키는 명령어

모듈

import 모듈

* 모듈.함수 🡪 함수 실행됨

import 모듈 as 이름 🡪 새로운 이름으로 모듈 접근 가능

from 모듈 import \*

* 함수 🡪 함수 실행됨 (import 모듈과 다른점이 이것)

from 모듈 import 함수들

* 불러온 함수들만 사용 가능

패키지

* 모듈의 집합

import 패키지.모듈

* 패키지의 모듈을 불러옴
* 패키지.모듈.~ 로 접근

from 패키지 import 모듈

from 패키지.모듈 import 클래스

\_\_init\_\_.py

\_\_all\_\_ = [“”] 🡪 “”사용해도 되는 모듈을 적어 넣음 (from 패키지 import \*로 모든 모듈을 불러올 수 없음, 공개하지 않는 모듈도 존재할 수 있기 때문, 그래서 \_\_all\_\_에 넣어줘야 함)

\_\_name\_\_ == “\_\_main\_\_”이면 해당 모듈에서 직접 명령어를 실행

아니면 외부에서 모듈을 호출하여 명령어를 실행 (과 같이도 할 수 있음)

1. 모듈 위치 아는 법

import inspect

import 모듈

print(inspect.getfile(모듈))

내장 함수 몇 가지 (built-in functions):

Type(a)

* 자료형을 나타냄

isinstance(x, int)

* x가 int형인지 확인

Id(a)

* 주소 값을 나타냄

a is b

* A와 B의 객체가 같은지 True와 False로 나타냄

Pass

* 아무 일도 하지 않고 건너감 (임시로 코드를 만들 때 주로 사용)

Import <module\_name>

* module을 가져옴

From <module\_name> import func1, func2

* module에서 func1과 func2를 불러옴

From <module\_name> import \*

* module에서 모두 꺼냄

zip(a, b)

* 튜플 a와 튜플 b를 묶음

super().\_\_init\_\_(a, b, c)

* super()는 상속을 받은 class의 기능을 활성화시키는 것인데
  + ex) a.\_\_init\_\_(self, a, b, c)라고 하는 것을 super().\_\_init\_\_(a, b, c)

로 해서 self를 안 써도 되는 것

dir()

* 어떤 객체를 넘겼을 때 그 객체가 어떤 변수 혹은 함수를 갖고 있는지 보여줌

ex) dir(random) 🡪 random으로 사용할 수 있는 함수 나열

globals()

* 전역 namespace를 볼 수 있음

locals()

* 지역 namespace를 볼 수 있음
  + 함수 밖에서 사용 시 globals()와 같은 효과

외장 함수 몇 가지:

import random

import os

import time

표준 입력 함수

print(~~~, end = “ “)

* end = “ “는 다음 print에 대해서 줄 바꿈 없이 이어 나가도록 해줌

print(a, b, sep = “ vs “

* a vs b (콤마 사이를 sep으로 구분지음)

a.ljust(정수), b.rjust(정수)

* a를 정수만큼 왼쪽 정렬
* b를 정수만큼 오른쪽 정렬

str.zfill(정수)

* 정수만큼 공간을 확보하고 str이 빈공간을 갖고 있으면 그만큼 0을 붙임
  + ex) 3개의 공간 확보 시 🡪 1 🡪 001

정규표현식

import re

p = re.compile(‘ab\*’)

m = re.match(string)

* 축약형

m = re.match(‘ab\*’, string)

Raw string 규칙

* 백슬래시 문제

정규식에서 는 [\\가](file:///\\가) 아닌 \\\\로 써야 백슬래시로 인식함

* + 이를 쉽게 하기 위해 밑을 사용

re.compile(r’\\문자열’)

컴파일된 객체 p가 갖는 메서드:

* match() : 문자열의 처음부터 정규식과 매치되는지 조사 (불참 등장 시 끝)
  + match객체 반환, 없으면 None 반환
* search() : 문자열 전체를 검색하여 정규식과 매치되는지 조사 (계속됨)
  + match객체 반환, 없으면 None 반환
* findall() : 정규식과 매치되는 모든 문자열을 리스트로 반환
* finditer() : 정규식과 매치되는 모든 문자열을 반복 가능한 객체로 반환
  + match 객체

match 객체가 갖는 메서드

* group() : 매치된 문자열을 반환
* start() : 매치된 문자열의 시작 인덱스를 반환
* end() : 매치된 문자열의 끝 인덱스를 반환
* span() : 매치된 문자열의 (시작, 끝)에 해당하는 튜플을 반환

컴파일 옵션:

* DOTALL 또는 S: \n을 포함하여 모든 문자를 매칭하도록 만듦

ex) re.compile(‘a.b’, re.DOTALL)

* IGNORECASE 또는 I: 대소문자 구별없이 매칭하도록 만듦

ex) re.compile(‘[a-z]+’, re.IGNORECASE)

* MULTILINE 또는 M: 메타 문자를 문자열의 각 줄마다 적용

ex) re.compile(‘^python\s\w+’, re.MULTILINE)

* VERBOSE 또는 X: 정규식을 주석 또는 줄 단위로 구분할 수 있도록 해줌

ex) re.compile(‘’’

[a-z]+ #a to z

……

‘’’), re.VERBOSE)

[ ] : 문자클래스; []안에 들어가는 것과 매칭시켜 해당하는 것이 있는지 체크

[abc]

[a-f]

[0-5]

^를 사용하면 반대의 것을 매칭시킴

자주 쓰이는 표현:

\d: 숫자와 매칭

\D: 숫자 아닌 것과 매칭

\s: 공백문자와 매칭 == [ \t\n\r\f\v]와 같음

\S: 공백문자 아닌 것과 매칭

\w: 문자+숫자와 매칭

\W: 문자+숫자가 아닌 문자와 매칭

. : \n을 제외한 모든 문자와 매칭

ex) a.b 🡪 a0b, acb, a b 등 매치

* [.]하면 . 매치

\* : 반복; 앞 문자가 0부터 계속 반복되는 글자 매칭

ex) ca\*t 🡪 ct, caaat, cat, caat…

+ : 반복; \*과는 다르게 1부터 반복되는 것을 매칭

ex) ca\*t 🡪 cat, caaat, caaaat… ct(x)

{m,n} : 반복; m부터 n까지 반복된 문자와 매칭

* {m} : m번 반복
* {m,} : m부터 쭉
* {,n} : n까지 쭉

ex) ca{2}t, ca{1,4}t, ca{2,}t, ca{,2}t

? : {0, 1}과 같음 [있어도 되고 없어도 된다는 뜻]

메타문자

* | : A 또는 B

ex) p = re.compile(‘Ben|Zach’)

* ^ : 맨 처음과 일치하는 문자열 매칭

ex) p = re.search(‘^Life’, ‘Life is too short’) Yes

ex) p = re.search(‘^Life’, ‘My Life’) No

* $ : ^와 반대로, 끝과 일치하는 문자열 매칭

ex) p = re.search(‘short$’, ‘Life is too short’)

* \A : ^과 같지만, MULTILINE 컴파일 방식을 택하면 각 줄이 아닌 전체 줄의 첫 문자열과 일치하는 것을 매칭
* \Z : $과 같지만, MULTILINE 컴파일 방식을 택하면 각 줄이 아닌 전체 줄의 끝 문자열과 일치하는 것을 매칭
* \b : 단어 구분자; 같은 단어이더라도 공백으로 구분이 되어 있는 문자를 매칭시킬 때 사용

ex) re.search(r‘\bclass\b’, ‘no class at all’) Yes

ex) re.search(r‘\bclass\b’, ‘declassified’) No

* + 주의: raw string 규칙을 따라야 함 🡪 \b가 파이썬에서는 백스페이스이기 때문
* \B : \b와는 반대로, 공백으로 구분되지 않은 문자만 매칭시킴

ex) re.search(r‘\bclass\b’, ‘no class at all’) No

ex) re.search(r‘\bclass\b’, ‘declassified’) Yes

그루핑 : 특정 패턴의 문자열이 반복될 때 매칭을 위해 그루핑 필요

* 보통 정해진 패턴 안의 특정 문자열을 뽑고 싶을 때 사용
* ( ) : 괄호를 사용해서 그루핑
* group메서드 🡪 group()에 group(2)와 같이 괄호의 순서로 인덱스를 달으면 특정 문자열 얻을 수 있음
* 재참조 🡪 (r’(\b\w+)\s+\1’) 🡪 \1은 그룹1을 재참조하겠다는 뜻 (다시 말해 재활용)
* 정규식 이름 짓기: (?P<그룹명>…)

ex) (?P<name>\w+)

m.group(“name”)

* + \w+의 패턴에 name이라는 이름을 붙여준 것
  + 재참조 🡪 (?P=name)

전방 탐색 : 검색 과정에는 포함이 되지만, 검색 결과에는 나오지 않게 (소비되지 않게 하는 법)

긍정형 전방 탐색: 정규식이 매치되는 조건이 통과되고 문자열은 소비하지 않음

* (?=…)

ex) (“.+(?=:)”)

부정형 전방 탐색: 정규식의 매치되지 않는 조건이 통과되고 문자열은 소비하지 않음

* (?!...)

ex) .\*[.](?!bat$|exe$).\*$

문자열 바꾸기 (치환) : 특정 문자(들)을 다른 문자로 표현 가능

* sub

ex) p = re.compile(‘(blue|red|yellow)’)

p.sub(‘color’, ‘blue and red socks’) 🡪 color and color socks

p.sub(‘color, ‘blue and red socks’, count=1) 🡪 color and red socks

* subn : 바뀐 문자와 치환된 개수를 튜플로 반환

ex) 위의 결괏값 🡪 (‘color and red socks’, 1)

* \g : 특정 구문을 참조하여 그 구문으로 상호작용함

ex) p.sub(“\g<그룹명2> \g<그룹명1>”, “park 010-1234-1234”) 🡪 010~ park

* + 그룹명1과 2의 순서를 바꿈 (참조번호여도 됨)
* sub(함수, 문자열)도 가능 🡪 문자열을 함수를 이용하여 바꾸는 등…

Greedy vs. Non-greedy

ex) <html><head><title>

에서 <.\*>를 사용하면 첫 ‘<’ 부터 끝 ‘>’까지 모두 가져옴

이를 방지하기 위해 <.\*?> 과 같이 ‘?’를 뒤에 붙여주면 non-greedy하게 탐욕스럽지 않은 방식으로 하나만 가져옴

* \*?, +?, ??, {m,n}? 등 사용 가능

팁

1. Prompt = “””

Wow it’s

Incredible!!

“””

와 같이 엔터가 포함 된 긴 문장의 문자열을 저장할 수 있다

1. Print(string[start:end:step])

콜론 두 개를 사용, **start부터 end까지 step만큼** 더하면서 그 인덱스를 출력

* + string[::-1]의 경우 반대로 스텝을 밟음

1. Print(f”이름: {name} 나이: {age}”)

* 파이썬 3.6부터 지원하는 “f-string”
  + 문자열 앞에 f를 작성함으로써 변수를 문자열에 대입함으로써 사용 가능

1. 문장 뒤에 \(역슬래쉬)를 써서 같은 줄에 있는 다음 문장을 다음 줄에 써도 같은 문장으로 취급되도록 할 수 있음 (줄이 너무 길어서 한 눈에 안 들어와서 사용)
2. input().split()을 써서 다수의 입력을 한 줄에 받게 할 수 있다. (각 문자를 공백을 갖고 구분)
3. ord(a) 🡪 아스키코드 a를 해당하는 번호 값으로 변경시킴
4. chr(a) 🡪 번호 a를 해당하는 아스키코드 값으로 변경시킴
5. while 리스트:를 할 시에 리스트의 요소가 없을 때까지 진행
6. 16진수 치려면 print(“%X”, % 변수)
7. 16진수로 바꾸려면 int(변수, 16)
8. from random import \* 🡪 난수모듈, random(): 베이스는 0~9, randrange(start, end), randint(start, end): start end 모두 포함
9. 파이썬은 call by value, call by reference가 아닌 call by assignment
   * 파이썬에는 reference counter라는 것이 존재하는데, 이것은 각 객체에 할당 된 변수의 개수를 세는 용도
   * 즉, 같은 객체를 갖게 되는 변수는 모두 주소가 같음. 새로 대입식을 쓰지 않는 한 주소는 계속 동일.
10. 파이썬은 return 값이 여러 개 일 수 있음
11. import sys 🡪 sys.stdin.readline()을 쓰는 게 일반 input()보다 실행속도 측면에서 우월함
12. deque라이브러리에서 popleft()를 쓰는 게 일반 pop(0)보다 실행속도 측면에서 우월함
13. ==은 값을 비교, is는 객체를 비교
    * 객체의 경우 배열 등의 객체의 경우 주소가 매번 다름
14. [‘a’ for i in range(3)]과 같이 즉시 리스트 자동생성 가능

웹 크롤링

웹 스크래핑