[프로그램 1-1]

import turtle

import random

myTurtle, tX, tY, tColor, tSize, tShape = [None] \* 6

shapeList = []

playerTurtles = []

swidth, sheight = 500, 500

if \_\_name\_\_ == "\_\_main\_\_" :

turtle.title('거북 리스트 활용')

turtle.setup(width = swidth + 50, height = sheight + 50)

turtle.screensize(swidth, sheight)

shapeList = turtle.getshapes()

for i in range(0, 100):

random.shuffle(shapeList)

myTurtle = turtle.Turtle(shapeList[0])

tX = random.randrange(-swidth / 2, swidth / 2)

tY = random.randrange(-sheight / 2, sheight / 2)

r = random.random(); g = random.random(); b = random.random()

tmpShape = random.choice(shapeList)

myTurtle = turtle.Turtle(tmpShape)

tSize = random.randrange(1, 3)

playerTurtles.append([myTurtle, tX, tY, tSize, r, g, b])

for tList in playerTurtles:

myTurtle = tList[0]

myTurtle.color((tList[4], tList[5], tList[6]))

myTurtle.pencolor((tList[4], tList[5], tList[6]))

myTurtle.turtlesize(tList[3])

myTurtle.goto(tList[1], tList[2])

turtle.done()

[프로그램 1-2]

foods={"떡볶이":"오뎅",

"짜장면":"단무지",

"라면":"김치",

"피자" : "피클",

"맥주":"땅콩",

"치킨":"치킨무",

"삼겹살":"상추"};

while (True):

myfood = input(str(list(foods.keys())) + "중 좋아하는 음식은?")

if myfood in foods:

print("<%s> 궁합 음식은 <%s>입니다." % (myfood, foods.get(myfood)))

elif myfood == "끝":

break;

else:

print("그런 음식이 없습니다. 확인해 보세요.")

[프로그램 2-1]

inStr, outStr="",""

count, i=0,0

inStr=input("문자열을 입력하세요 : ")

count = len(inStr)

for i in range(0, count) :

outStr+= inStr[count - (i + 1)]

print("내용을 거꾸로 출력 --> %s" % outStr)

[프로그램 2-2]

import turtle

import random

from tkinter.simpledialog import\*

inStr =''

swidth, sheight=300,300

tX, tY, txtSize = [0]\*3

turtle.title('거북이 글자쓰기')

turtle.shape('turtle')

turtle.setup(width = swidth + 50, height = sheight + 50)

turtle.screensize(swidth, sheight)

turtle.penup()

inStr = askstring('문자열 입력', '거북이 쓸 문자열을 입력')

for ch in inStr :

tX = random.randrange(-swidth / 2, swidth / 2)

tY = random.randrange (-sheight / 2, sheight / 2)

r=random.random(); g=random.random(); b=random.random()

txtSize = random.randrange(10,50)

turtle.goto(tX,tY)

turtle.pencolor((r, g, b))

turtle.write(ch, font=('맑은고딕', txtSize, 'bold'))

turtle.done()

[3-1]

from typing import Any, Sequence

def seq\_search(a: Sequence, key: Any) -> int:

"""시퀀스 a에서 key와 값이 같은 원소를 선형 검색(while 문)"""

i = 0;

while True:

if i == len(a):

return -len # 검색에 실패하여 -1을 반환

if a[i] == key:

return i # 검색에 성공하여 현재 검사한 배열의 인덱스를 반환

i += 1

if \_\_name\_\_ == '\_\_main\_\_':

num = int(input('원소 수를 입력하세요.: ')) # num값을 입력받음

x = [None] \* num # 원소 수가 num인 배열을 생성

for i in range(num):

x[i] = int(input(f'x[{i}]: '))

ky = int(input('검색할 값을 입력하세요.: ')) # 검색할 키 ky를 입력받음

idx = seq\_search(x, ky) # ky와 값이 같은 원소를 x에서 검색

if idx == -1:

print('검색값을 갖는 원소가 존재하지 않습니다.')

else:

print(f'검색값은 x[{idx}]에 있습니다.')

[3-2]

from typing import Any, Sequence

def seq\_search(a: Sequence, key: Any) -> int:

"""시퀀스 a에서 key와 값이 같은 원소를 선형 검색(for 문)"""

for i in range(len(a)):

if a[i] == key:

return i # 검색 성공(인덱스를 반환)

return -1 # 검색 실패(-1을 반환)

if \_\_name\_\_ == '\_\_main\_\_':

num = int(input('원소 수를 입력하세요.: ')) # num값을 입력받음

x = [None] \* num # 원소 수가 num인 배열을 생성

for i in range(num):

x[i] = int(input(f'x[{i}]: '))

ky = int(input('검색할 값을 입력하세요.: ')) # 검색할 키 ky를 입력받음

idx = seq\_search(x, ky) # ky와 값이 같은 원소를 x에서 검색

if idx == -1:

print('검색값을 갖는 원소가 존재하지 않습니다.')

else:

print(f'검색값은 x[{idx}]에 있습니다.')

[3-3]

from typing import Any, Sequence

import copy

def seq\_search(seq: Sequence, key: Any) -> int:

"""시퀀스 seq에서 key와 일치하는 원소를 선형 검색(보초법)"""

a = copy.deepcopy(seq) # seq를 복사

a.append(key) # 보초 key를 추가

i = 0

while True:

if a[i] == key:

break # 검색에 성공하면 while 문을 종료

i += 1

return -1 if i == len(seq) else i

if \_\_name\_\_ == '\_\_main\_\_':

num = int(input('원소 수를 입력하세요.: ')) # num값을 입력받음

x = [None] \* num # 원소 수가 num인 배열을 생성

for i in range(num):

x[i] = int(input(f'x[{i}]: '))

ky = int(input('검색할 값을 입력하세요.: ')) # 검색할 키 ky를 입력받음

idx = seq\_search(x, ky) # ky와 값이 같은 원소를 x에서 검색

if idx == -1:

print('검색값을 갖는 원소가 존재하지 않습니다.')

else:

print(f'검색값은 x[{idx}]에 있습니다.')

[3-4]

from typing import Any, Sequence

def bin\_search(a: Sequence, key: Any) -> int:

"""시퀀스 a에서 key와 일치하는 원소를 이진 검색"""

pl = 0 # 검색 범위 맨 앞 원소의 인덱스

pr = len(a) - 1 # 검색 범위 맨 끝 원소의 인덱스

while True:

pc = (pl + pr) // 2 # 중앙 원소의 인덱스

if a[pc] == key:

return pc # 검색 성공

elif a[pc] < key:

pl = pc + 1 # 검색 범위를 뒤쪽 절반으로 좁힘

else:

pr = pc - 1 # 검색 범위를 앞쪽 절반으로 좁힘

if pl > pr:

break

return -1 # 검색 실패

if \_\_name\_\_ == '\_\_main\_\_':

num = int(input('원소 수를 입력하세요.: '))

x = [None] \* num # 원소 수가 num인 배열을 생성

print('배열 데이터를 오름차순으로 입력하세요.')

x[0] = int(input('x[0]: '))

for i in range(1, num):

while True:

x[i] = int(input(f'x[{i}]: '))

if x[i] >= x[i - 1]: # 바로 직전에 입력한 원솟값보다 큰 값을 입력

break

ky = int(input('검색할 값을 입력하세요,: ')) # 검색할 키값 ky를 입력

idx = bin\_search(x, ky) # ky값과 같은 원소를 x에서 검색

if idx == -1:

print('검색값을 갖는 원소가 존재하지 않습니다.')

else:

print(f'검색값은 x[{idx}]에 있습니다.')

[3-5]

from \_\_future\_\_ import annotations

from typing import Any, Type

import hashlib

class Node:

"""해시를 구성하는 노드"""

def \_\_init\_\_(self, key: Any, value:Any, next: Node) -> None:

"""초기화"""

self.key = key # 키

self.value = value # 값

self.next = next # 뒤쪽 노드를 참조

class ChainedHash:

"""체인법으로 해시 클래스 구현"""

def \_\_init\_\_(self, capacity: int) -> None:

"""초기화"""

self.capacity = capacity # 해시 테이블의 크기를 지정

self.table = [None] \* self.capacity # 해시 테이블(리스트)을 선언

def hash\_value(self, key: Any) -> int:

"""해시값을 구함"""

if isinstance(key, int):

return key % self.capacity

return(int(hashlib.sha256(str(key).encode()).hexdigest(), 16) % self.capacity)

def search(self, key: Any) -> Any:

"""키가 key인 원소를 검색하여 값을 반환"""

hash = self.hash\_value(key) # 검색하는 키의 해시값

p = self.table[hash] # 노드를 주목

while p is not None:

if p.key == key:

return p.value # 검색 성공

p = p.next # 뒤쪽 노드를 주목

return None # 검색 실패

def add(self, key: Any, value: Any) -> bool:

"""키가 key이고 값이 value인 원소를 추가"""

hash = self.hash\_value(key) # 추가하는 key의 해시값

p = self.table[hash] # 노드를 주목

while p is not None:

if p.key == key:

return False # 추가 실패

p = p.next # 뒤쪽 노드를 주목

temp = Node(key, value, self.table[hash])

self.table[hash] = temp # 노드를 추가

return True # 추가 성공

def remove(self, key: Any) -> bool:

"""키가 key인 원소를 삭제"""

hash = self.hash\_value(key) # 삭제할 key의 해시값

p = self.table[hash] # 노드를 주목

pp = None # 바로 앞의 노드를 주목

while p is not None:

if p.key == key: # key를 발견하면 아래를 실행

if pp is None:

self.table[hash] = p.next

else:

pp.next = p.next

return True # key 삭제 성공

pp = p

p = p.next # 뒤쪽 노드를 주목

return False # 삭제 실패(key가 존재하지 않음)

def dump(self) -> None:

"""해시 테이블을 덤프"""

for i in range(self.capacity):

p = self.table[i]

print(i, end='')

while p is not None:

print(f' -> {p.key} ({p.value})', end='')

p = p.next

print()

[3-6]

from enum import Enum

from chained\_hash import ChainedHash

Menu = Enum('Menu', ['추가', '삭제', '검색', '덤프', '종료']) # 메뉴를 선언

def select\_menu() -> Menu:

"""메뉴 선택"""

s = [f'({m.value}){m.name}' for m in Menu]

while True:

print(\*s, sep = ' ', end='')

n = int(input(': '))

if 1 <= n <= len(Menu):

return Menu(n)

hash = ChainedHash(13) #크기가 13인 해시 테이블을 생성

while True:

menu = select\_menu() # 메뉴 선택

if menu == Menu.추가: # cnrk

key = int(input('추가할 키를 입력하세요.: '))

val = input('추가할 값을 입력하세요.: ')

if not hash.add(key, val):

print('추가를 실패했습니다!')

elif menu == Menu.삭제: # 삭제

key = int(input('삭제할 키를 입력하세요.: '))

if not hash.remove(key):

print('삭제를 실패했습니다!')

elif menu == Menu.검색: # 검색

key = int(input('검색할 키를 입력하세요.: '))

t = hash.search(key)

if t is not None:

print(f'검색한 키를 갖는 값은 {t}입니다.')

else:

print('검색할 데이터가 없습니다.')

elif menu == Menu.덤프: # 덤프

hash.dump()

else: # 종료

break