파이썬

상명대학교 융합공과대학
지능•데이터융합학부
휴먼지능정보공학전공
dkim@smu.ac.kr

강의개요

- 강의소개 및 프로그래밍 개념
  - 프로그래밍과 컴퓨팅사고력 소개
  - 프로그래밍 맛보기
- 변수, 자료형, 연산, 함수
  - 코딩과 기초실습
- 조건문, 연산자
  - 코딩과 기초실습
- 반복문
  - 코딩과 기초실습
- 함수, 매개변수
  - 코딩과 기초실습
- 중간고사

강의개요

- 자료형, 리스트
  - 코딩과 기초실습
- 자료형, 튜플
  - 코딩과 기초실습
- 자료형, 딕션너리
  - 코딩과 기초실습
- 실습예제
  - 코딩과 기초실습
- 파일읽고 쓰기
  - 코딩과 기초실습
- 객체지향 프로그래밍
  - 코딩과 기초실습
- 기말고사

프로그래밍 이해하기

- 프로그래밍
  - 프로그래밍(programming)은 프로그램을 작성하는 것
  - 프로그램을 작성하는 사람을 프로그래머(programmer)
  - 프로그래밍 작성 과정이 프로그래밍 또는 코딩(coding)
  - 컴퓨터가 이해할 수 있는 규칙에 따라 프로그램 수행절차를 프로그래밍 언어로 작성하는 것

- 프로그래밍언어
  - 컴퓨터 시스템을 동작시키는 소프트웨어를 작성하기 위한 형식언어
  - 컴퓨터를 이용하여 특정 문제를 해결하기 위한 프로그램을 작성하기 위해 사용되는 언어
  - 컴퓨터 소프트웨어를 만드는 언어

프로그래밍 이해하기

- 코딩
  - ‘컴퓨터 프로그램을 수행하는 절차를 적어 둔 명령어들인 코드(code)를 작성하는 행위’
  - 코딩은 ‘문제해결을 위한 절차와 과정을 설정하고, 그것을 실행 가능한 프로그램으로 작성하는 일’
  - 코딩의 궁극적 목표는 주어진 문제를 제대로 해결하는 일
  - 코딩에 앞서 문제해결을 위한 방법을 먼저 구상
    - 효율적인 코딩은 먼저 알고리즘(algorithm)부터 구상
    - 알고리즘은 ‘어떤 작업을 수행하는데 있어 적합한 절차와 과정’

프로그래밍 기본요소

- 데이터(자료): 프로그램을 운용할 수 있는 형태로 기호화 숫자화 한 자료(데이터)
  - 숫자, 문자
- 변수: 데이터(자료)를 저장하는 공간 (이름, 형태가 있음)
  - 정수형,실수형,문자,문자열(str)형,리스트(list)형,불(bool)형,튜플(tuple),집합(set)형, 사전(dict)형 등
- 상수: 항상 같은 값을 가지는 수나 문자 데이터(자료) 자체
  - 숫자, 문자

프로그래밍 기본요소

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 정수형,실수형,문자,문자열(str)형,리스트(list)형,불(bool)형,튜플(tuple),집합(set)형, 사전(dict)형 등

- a=10
  - a(변수, 정수형) = (대입연산자) 10(데이터, 자료, 상수)

프로그래밍 기본요소

- 자료구조: 데이터를 조직(생성), 저장, 표현하는데 요구되는 방식과 이를 구현하는데 필요한 알고리즘에 관한 이론
  - 저장, 탐색, 삭제
- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 대입연산, 산술연산, 증감연산, 관계연산, 논리연산, 비교연산, 삼항연산
- 명령문: 프로그램이 특정 작업을 수행하도록 프로그래밍 언어로 작성하는 명령
  - 선언문, 대입문, 함수(호출)문, 반복문(for, while), 조건문(if~else, if~elseif, switch case), 분기문(break, continue)

프로그래밍 기본요소

- 함수: 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 코드 모음
  - 내장함수(매개변수, 반환 값), 사용자정의함수(매개변수, 반환 값), 라이브러리(매개변수, 반환 값)

  - def 함수이름(매개변수) :
    실행하고자 하는 함수내용
    (변수, 상수, 연산자, 대입문, 선언문, 반복문, 조건문, 분기문, 함수호출 등)

  - 함수이름(매개변수)

프로그래밍 문법

- 변수: 데이터를 저장하는 공간
  - 다른 변수와 구별되게 고유한 이름을 붙인 것을 명칭
  - 이름을 붙이는 규칙(명칭 규칙)
    - 키워드 사용 안됨 (if, else, for, while, break, switch, case,True, False, and, or, not, def return)
    - 대소문자 구분
    - 주로 알파벳, 숫자, 밑줄문자로 구성 (공백, +,- 기호 사용 안됨)
    - 첫 글자로 숫자 사용 안됨
  - 변수 사용
    - 파이썬의 경우 변수는 별도의 타입을 지정하지 않음

```
>>>testNum=100
>>>print(testNum)
100
>>>testStr="대한민국"
>>>print(testStr)
대한민국
```

프로그래밍 문법

- 자료형(타입) : 데이터(자료)를 표현하는 방법(형태(타입)가 있음)
  - 수치형
    - 정수형: 가장 간단한 수치형 (소수점이 없음)

      ```
      >>>a=10
      >>>print(a)
      10
      ```

    - 실수형: 소수점이 있는 수치형

      ```
      >>>a=10.2e2
      >>>print(a)
      1020.0
      ```

    - 복소수형: 실수부와 허수부가 있는 수치형

      ```
      >>>a=1+2j
      >>>b=2+4j
      >>>print(a+b)
      (3+6j)
      ```

프로그래밍 문법

- 자료형(타입) : 데이터(자료)를 표현하는 방법(형태(타입)가 있음)
  - 문자열: 문자의 나열로써 따옴표 안에 작성
    - 한글, 영문, 한자 모두 표현 가능
    - 따옴표 안에 작성하면 숫자로 문자열로 처리

```
>>> a="1"
>>> b="2"
>>> print(a+b)
12
```

프로그래밍 문법

- 자료형(타입) : 데이터(자료)를 표현하는 방법(형태(타입)가 있음)
  - 문자열: 문자의 나열로써 따옴표 안에 작성
    - 긴 문자열을 표현할 때는 따옴표 세 개를 사용

```
a="동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세
무궁화 삼천리 화려 강산 대한 사람 대한으로 길이 보전 하세"
>>>print(a)
동해물과 백두산이 마르고 닳도록 하느님이 보우하사 우리나라 만세 무궁화 삼천리 화려 강산 대한 사람 대한으로 길이 보전 하세
>>>b="""남산 위에 저 소나무 철갑을 두른 듯 바람 서리 불변함은 우리 기상일세
무궁화 삼천리 화려 강산 대한 사람 대한으로 길이 보전 하세"""
>>>print(b)
남산 위에 저 소나무 철갑을 두른 듯 바람 서리 불변함은 우리 기상일세
무궁화 삼천리 화려 강산 대한 사람 대한으로 길이 보전 하세
```

프로그래밍 문법

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 대입연산: 오른쪽에 있는 값을 왼쪽의 변수에 저장할 때 사용

```
>>>a=10
>>>print(a)
10
```

  - 산술연산: 숫자형 연산할 때 사용(사칙연산, 거듭제곱, 정수나누기, 나머지 연산)

```
>>>a=10
>>>b=a**2
100
>>>a=7
>>>b=a%2
```

프로그래밍 문법

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 복합대입연산: 오른쪽에 있는 변수와 값을 왼쪽 변수에 저장할 때 사용

```
>>>a=10
>>>a=a+20
>>>print(a)
30
```

프로그래밍 문법

- 자료형(타입)변환: 데이터(자료)를 표현하는 방법을 변환
  - 문자열연산

    >>>a="대한"
    >>>b="민국"
    >>>c="만세"
    >>>print((a+b))+(c*3))
    대한민국만세만세만세
  - 정수형 변환: 문자열을 숫자(정수)로 변환하기 위해 int()함수를 사용

    >>>a=10
    >>>b="20"
    print(a+(int(b))
    30

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 정수형,실수형,복소수형
  - 문자, 문자열(str)형
  - 불(bool)형
  - 리스트(list)형
  - 튜플(tuple)형
  - 집합(set)형
  - 사전(dict)형

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 정수형, 실수형, 복소수형
    - 정수형: 가장 간단한 수치형 (소수점이 없음)
      - a=20
      - b=20
      - c = a+ b
      - print(c) → 40

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 정수형, 실수형, 복소수형
    - 실수형: 소수점이 있는 수치형
      - a=20.34
      - b=20.54
      - c=2044.22e-2
      - d=a+b
      - e=b+c
      - print(c) → 20.4422
      - print(d) → 40.879999999999995
      - print(e) → 40.9822
      - print('%.4f' % d) → 40.8800 ## print("%출력원하는 자리수f"%출력값)
      - print(round(d)) → 41
      - print(round(d, 1) → 40.9

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 문자, 문자열(str)형 한 개의 문자, 여러 개의 문자열을 표현하고 처리
    - a= '상' → 문자
    - b = "상명" → 문자열
    - c= 상명대학교 → 오류
    - d = "우리는 '상명대학교' 학생입니다"
    - e = 100 +"100" → 오류
    - f = str(100) +"100" → 100100

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 문자, 문자열(str)형: 한 개의 문자, 여러 개의 문자열을 표현하고 처리
    - g = input(“이름을 입력하세요“) → 상명대
    - h = print(“안녕하세요”+g+”님 반갑습니다“) → 안녕하세요 상명대님 반갑습니다.

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 불(bool)형: 참과 거짓을 표현하고 처리
    - a = True → ##1
    - b = False → ##0
    - c = a + b → 1
    - d = a - b →1
    - e = a * b → 1
    - f = a / b →##오류 (나누기 연산)
    - a = bool(1) → True
    - b=bool(0) → False

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 리스트(list)형: 순서가 있는 수정가능한 데이터의 집합
    - a = [1,3,5]
    - b = [2,4,6]
    - c = a + b
    - print(c) → [1,3,5,2,4,6]
    - a = [‘상’,’명‘]
    - b = [‘대’, ‘학‘, ‘교‘]
    - c = a + b
    - print(c) → [‘상’,’명’,’대’,’학’,’교’]
    - a =[“상명”]
    - b=[“대학 “]
    - c=[“교”]
    - print(a+b+c) → [“상명”,”대학”,”교”]

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 튜플(tuple)형: 순서가 있는 수정 불가능한 데이터의 집합
    - 데이터 접근, 삭제 가능, 추가 불가능
    - ( )괄호로 작성되어지며, 내부 원소는 ,로 구분
    - 튜플 이름 = (요소1, 요소2, 요소3,…)
    - 튜플 더하기 및 (정수)곱하기(반복) 연산 가능
    - a = (1,2,3)
    - b = (‘가’,’나’,’다’)
    - c = a + b
    - print(c) → (1,2,3, ‘가‘, ‘나’ ,’ 다’)

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 집합(set)형: 순서가 없고 중복을 허용하지 않는 데이터의 집합
  - a = set([1,2,3,4,5])
  - b = set([1,3,5,6])
  - c = a.intersection(b)
  - print(c) → {1,3,5}
  - d = a.union(b)
  - print(d) → {1, 2, 3, 4, 5, 6}

자료형

- 자료형: 데이터(자료)를 표현하는 방법(형태가 있음)
  - 사전(dict)형: 쌍이 있는 수정 가능한 데이터의 집합
    - 접근, 추가, 삭제 가능
    - 중괄호({ })로 묶여 있으며 키와 값의 쌍으로 이루어지고 내부 원소는, 로 구분
    - 딕셔너리 이름= {키1:값1, 키2:값2, 키3:값3,…}
    - a = {1:”가”, 2:”나”, 3:”다”, 4:”라”, 5:”마”}
    - print(a) → {1: '가', 2: '나', 3: '다', 4: '라', 5: '마'}

프로그래밍 기본요소

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 대입연산, 산술연산, 증감연산, 관계연산, 논리연산

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 대입연산
  - 산술연산
  - 증감연산
  - 관계연산
  - 논리연산

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 대입연산 (=)
    - a = 10
    - b = “상명”
    - d = [1,2,3]
    - e = print(a)

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 산술연산 (+, -, *, /)
    - a = 1 + 1 ➔ 2
    - b = a + 1 ➔ 3
    - a = 1 - 1 ➔ 0
    - b = a - 1 ➔ -1
    - a = 1 * 1 ➔ 1
    - b = a * 1 ➔ 1
    - a = 1 / 1 ➔ 1
    - b = a / 1 ➔ 1

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 증감연산 (+=, -=)
    - a = 1
    - a += 1
    - print(a) → 2
    - a -= 1
    - print(a) → 1

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 관계연산 (> : 크다, >= : 크거나 같다, < : 작다, <= : 작거나 같다, == : 같다,!= : 같지 않다)
    - a = 1
    - b = 1
    - print(a > b) → False
    - print(a >= b) → True
    - print(a < b) → False
    - print(a <= b) → True
    - print(a==b) → True
    - print(a != b) → False

연산자

- 연산자: 프로그램의 산술식이나 연산식을 표현하고 처리하기 위해 제공되는 기호
  - 논리연산 (and, or, not)
    - a = True
    - b = False
    - print(a and b) → False
    - print(a or b) → True
    - print(a and not b) → True

프로그래밍 기본요소

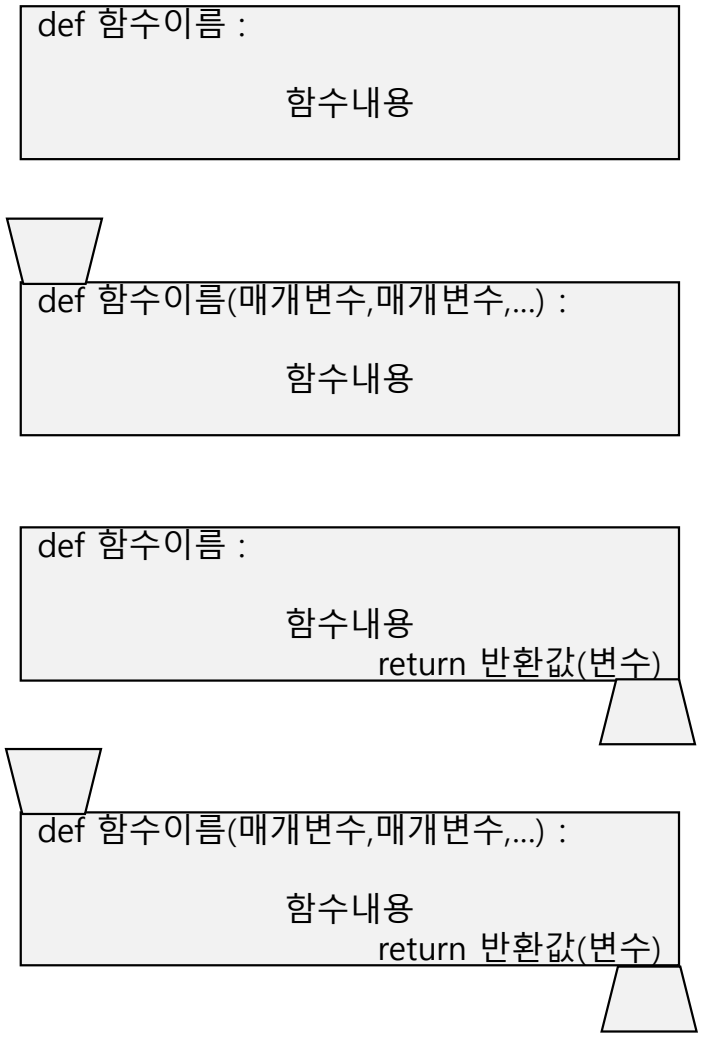
- 함수: 하나의 특별한 목적의 작업을 수행하기 위해 독립적으로 설계된 코드 모음
  - 내장함수(매개변수, 반환 값), 사용자정의함수(매개변수, 반환 값), 라이브러리(매개변수, 반환 값)

  - def 함수이름(매개변수) :
    실행하고자 하는 함수내용
    (변수, 상수, 연산자, 대입문, 선언문, 반복문, 조건문, 분기문, 함수호출 등)

  - 함수이름(매개변수)

함수

- 함수형태
  - 기본형
    - 매개변수, 반환값(변수)이 없는 형태
  - 매개변수형
    - 매개변수만 있는 형태
  - 반환형
    - 반환값(변수)만 있는 형태, return
  - 매개변수, 반환형
    - 매개변수, 반환값(변수)가 있는 형태, return



함수

- 함수종류
  - 내장함수: 파이썬에서 제공하는(포함되어 있는) 함수
    - 함수호출, 매개변수, 반환값(변수) 사용
    - import 키워드 사용하지 않음
      - 문자열출력함수: print("출력") → 출력
      - 입력함수: mytext = input("문자를 입력하세요") → 안녕하세요
      - 절대값함수: a=abs(-10) → 10
      - 문자열실행함수: eval("1+2") → 3
      - 16진수함수: a = hex(3) → 0x3
      - 정수변환함수: a = int(3.4) → 3
      - 배열길이반환함수: a = len([1,2,3]) → 3
      - 최대값반환함수 a= max([1,2,3]) → 3
      - 최솟값반환함수 a= min([1,2,3]) → 1

함수

- 함수종류
  - 내장함수: 파이썬에서 제공하는(포함되어 있는) 함수
    - 함수호출, 매개변수, 반환값(변수) 사용
    - import 키워드 사용하지 않음
      - 파일함수: f = open(“mytest.txt”, “r”) → 파일
      - 리스트함수: list((1,2,3)) → [1,2,3]
      - 제곱함수: pow(2,4) → 16
      - 반올림함수: round(4.6) → 5
      - 자료형반환함수: type(“abc”) → <class ‘str’>

함수

- 함수종류
  - 사용자정의함수: 사용자가 특정 작업을 수행하기 위해 만든 함수
    - 함수 이름 앞에 def 키워드를 사용
    - 함수 이름 다음 () 안에 매개변수 사용
    - () 다음에 : 붙임
    - 함수내용 작성
    - return 키워드 사용하여 반환값(변수) 지정
    - 재귀함수: 함수내용 안에 자기자신 함수(호출)를 사용한 함수

```
def MyHello(count):
    if count == 0:  # 종료 조건을 만듦. count가 0이면 다시 hello 함수를 호출하지 않고 끝냄
        return
    print("Hello World", count)
    count -= 1     # count를 1 감소시킨 뒤
    MyHello(count)  # 다시 Myhello에 넣음
MyHello(2)   # hello 함수 호출
```

→

```
Hello World, 2
Hello World, 1
```

함수

- 함수종류
  - 외장함수: 특별한 기능이 있는 모듈(함수의 집합)안에 있는 특정 함수를 사용
    - 함수를 포함시키기 위해 import 키워드를 사용
    - 시간모듈

```
import time
time.localtime() → time.struct_time(tm_year=2020, tm_mon=4, tm_mday=19, tm_hour=20, tm_min=20, tm_sec=40, tm_wday=6, tm_yday=110, tm_isdst=0):
# tm_wday는 요일(월요일~일요일, 0~6), tm_yday는 1월 1일부터 경과한 일수, tm_isdst는 서머타임 여부


import time
for i in range(10):
    print(i)
    time.sleep(2) → 2초 간격으로 0부터 9까지 숫자 출력
```

조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장

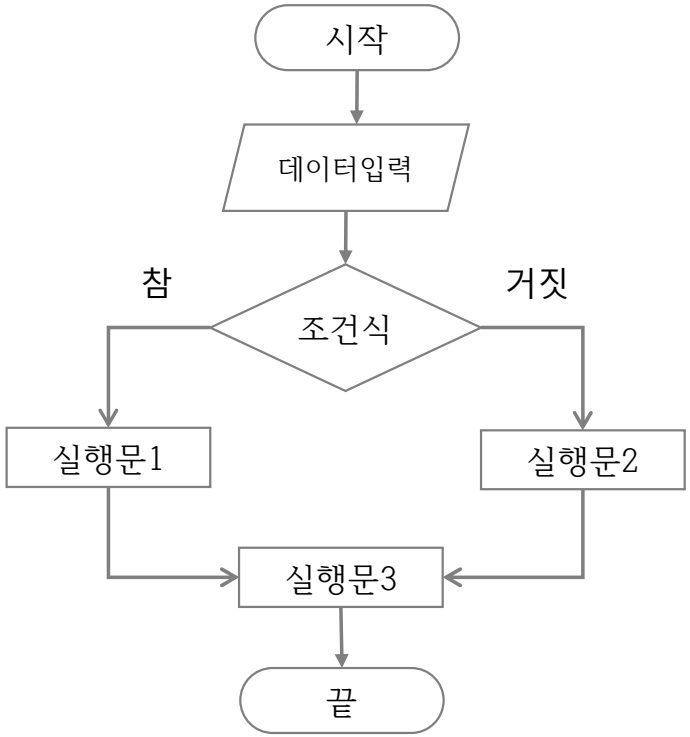
if 조건식 1 :
    조건식 1이 참인 경우 실행할 문장 1 (명령문, 함수)

else :
    조건식 1이 거짓인 경우 실행할 문장 2 (명령문, 함수)

if 조건문이 종료되고 실행할 문장 3 (명령문, 함수)

조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장



조건문

- 조건식: 참과 거짓을 구분할 수 있는 식
  - 관계연산자, 논리연산자, 포함연산자 사용
    - 대입연산자: a=0 (조건식이 아님, 항상 참)
    - 관계연산자:
      - a == b: a, b가 같으면 참(True) , ex) a=1, b=1, print(a==b) → True
      - != : a, b가 같지않으면 참(True)
      - a < b : a가 작으면 참(True)
      - a > b : a가 크면 참(True)
      - a <= b: a가 작거나 같으면 참(True)
      - a >= b: a가 크거나 같으면 참(True)
    - 논리연산자:
      - a or b : a와 b 둘 중에 하나만 참이면 참, ex) a=true, b=true, print(a or b) → True
      - a and b : a와 b 둘 다(모두) 참이어야 참 ex) a=true, b=true, print(a and b) → True
      - not a : a가 거짓이면 참

조건문

- 조건식: 참과 거짓을 구분할 수 있는 식
    - 포함연산자:
        - in : 문자열, 리스트, 튜플, 집합 등의 자료형에서 찾고자 하는 값(문자)이 포함되어 있으면 참
          a="상명", b="상명대학교", print(a in b) ##→ True
          a=10, b=[10,20,30,40], print(a in b) ## → True
          a=50, b=(10,20,30,40), print(a in b) ## → False
          a="상명", b={"상명대학교","융합공과대학","경영경제대학","계당교양교육원"}, print(a in b) ## → False

        - not in : 문자열, 리스트, 튜플, 집합 등의 자료형에서 찾고자 하는 값(문자)이 포함되어 있지 않으면 참
          a="상명", b="상명대학교", print(a not in b) ## → False
          a=10, b=[10,20,30,40], print(a not in b) ## → False
          a=50, b=(10,20,30,40), print(a not in b) ## → True
          a="상명", b={"상명대학교","융합공과대학","경영경제대학","계당교양교육원"}, print(a not in b) ## → True

조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장

if 조건식 1 :
    조건식 1이 참인 경우 실행할 문장 1(명령문, 함수)

elif 조건식 2:
    조건식 2가 참인 경우 실행할 문장 2(명령문, 함수)

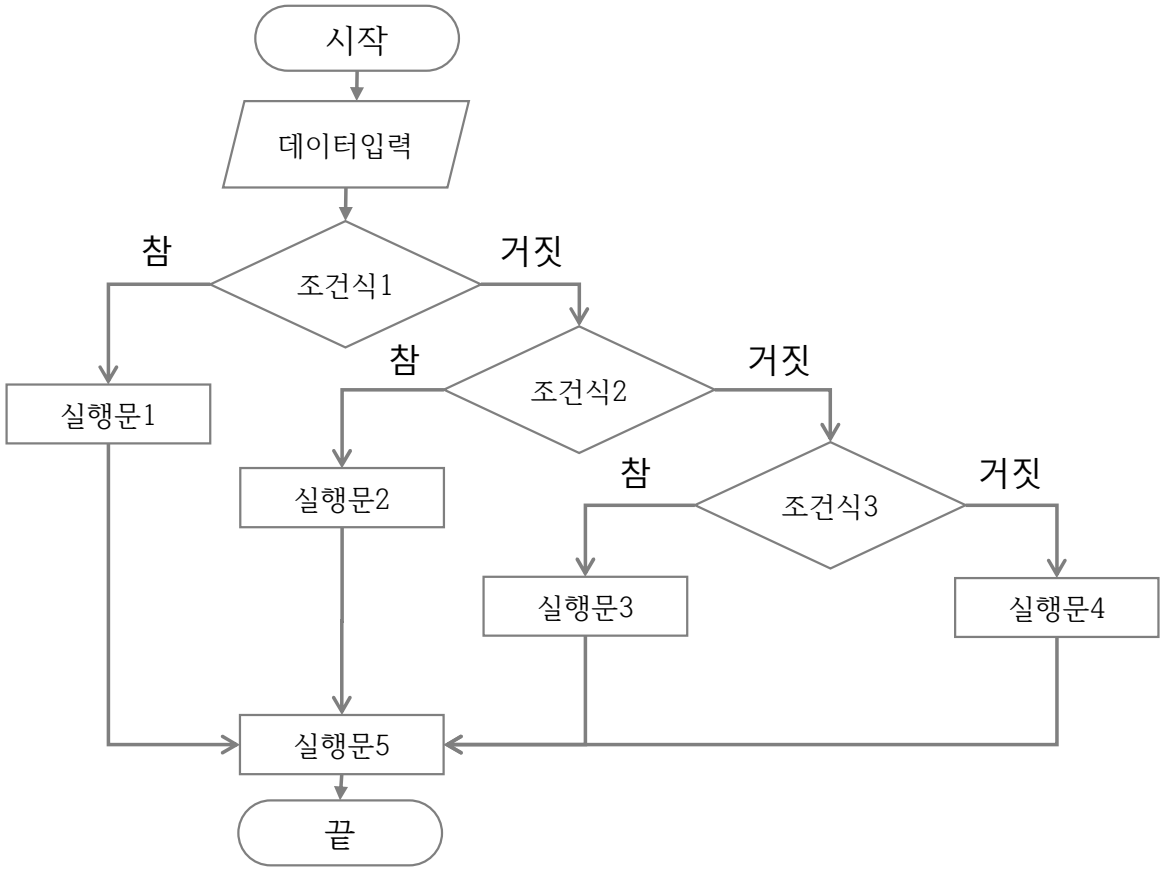
elif 조건식 3:
    조건식 3이 참인 경우 실행할 문장 3(명령문, 함수)

else :
    조건식 1,2,3이 거짓인 경우 실행할 문장 4 (명령문, 함수)

if 조건문이 종료되고 실행할 문장5 (명령문, 함수)

조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장



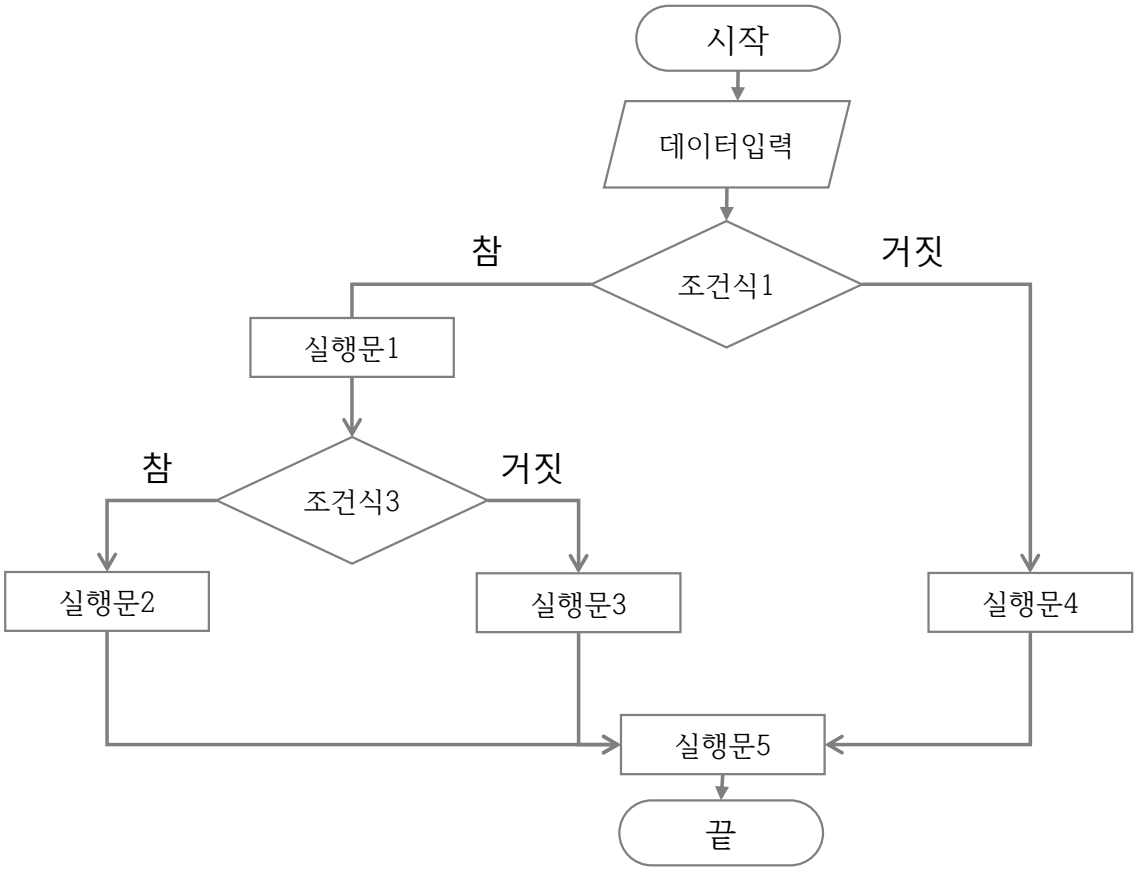
조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장

```
if 조건식 1 :
    조건식 1이 참인 경우 실행할 문장 1(명령문, 함수)

    if 조건식 2:
        조건식 1, 2가 참인 경우 실행할 문장 2(명령문, 함수)

    else 조건식 3:
        조건식 1이 참이고 조건식 2가 거짓인 경우 실행할 문장 3(명령문, 함수)
else :
    조건식 1이 거짓인 경우 실행할 문장 4(명령문, 함수)

if 조건문이 종료되고 실행할 문장5 (명령문, 함수)
```

조건문

- 조건문: 조건식이 참인 경우 실행하는 문장(명령문)과 거짓인 경우 실행하는 문장(명령문)을 구분하여 실행하도록 만든 문장



반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

```
num=1 ##→ 초기식
sum=0 ##→ 초기식
while num <=10 : ##→ 조건식
    sum = sum + num ##→ 실행문장 1
    num = num + 1 ##→ 실행문장 2
print("합계",sum) ##→ 조건식 만족 조건이 종료되는 경우 실행 문장 3
```

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

초기식
for 제어변수 in 컬렉션 : → 컬렉션은 문자열, 리스트, 튜플, range()함수가 해당
    조건식을 만족하는 동안 실행할 문장 1(명령문, 함수)

조건식 만족 조건이 종료되는 경우 실행할 문장 2(명령문, 함수)

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

for num in [0,1,2] : ##→ 컬렉션은 문자열, 리스트, 튜플, range()함수가 해당
    print(num, “회 출력합니다”)

print(“반복문이 종료됩니다”)
→
0회 출력합니다
1회 출력합니다
2회 출력합니다
반복문이 종료됩니다

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

for txt in “상명대학교” : ##→ 컬렉션은 문자열, 리스트, 튜플, range()함수가 해당
    print(txt, “이(가) 출력됩니다”)

print(“반복문이 종료됩니다”)
→
상 이(가) 출력됩니다
명 이(가) 출력됩니다
대 이(가) 출력됩니다
학 이(가) 출력됩니다
교 이(가) 출력됩니다
반복문이 종료됩니다

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

초기식
for 제어변수 in range(조건식1: 시작값, 끝값+1, 증가값)
    조건식1을 만족하는 동안 실행할 문장 1(명령문, 함수)

조건식1 만족 조건이 종료되는 경우 실행할 문장 2(명령문, 함수)

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

```
sum =0 ## →초기식
for num in range(1, 11, 1) : ## → 시작값은 1, 끝값은 10,  증가값은 1
    sum = sum + num
print(“합계” ,sum)
print(“반복문이 종료됩니다”)
## →
합계 55
반복문이 종료됩니다
```

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장

```
sum =0 ## → 초기식
for num in range(1, 11, 1) : ## → 시작값은 1, 끝값은 10,  증가값은 1
    sum = sum + num
    print("합계" ,sum)
print("반복문이 종료됩니다")
## →
합계 1
합계 3
합계 6
합계 10
합계 15
합계 21
합계 28
합계 36
합계 45
합계 55
반복문이 종료됩니다
```

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - 제어변수 활용: 반복 횟수나 끝낼 시점을 결정하기 위해 제어변수를 사용

for num in range(1,6,1) : ## → 시작값1, 끝값5, 증가값1
    print("*" * 10, end="") ## → end=""출력의 끝에 공백을 넣음 ##→ ********** 10개출력
    print(num *10, end="") ## → num (횟수) 출력
print("반복문이 종료됩니다")
##→
**********10 cm**********20 cm**********30 cm**********40 cm**********50 cm
반복문이 종료됩니다

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - break: 반복문을 종료하는 명령, 반복구문을 탈출

```
sum =0 ## → 초기식
for num in range(11) : ## → 시작값은 1, 끝값은 10,  증가값은 1
    sum = sum + num
    if num == 5 :
        print("5까지 합계" ,sum)
        break
print("전체합계" ,sum)
print("반복문이 종료됩니다")
## →
5까지 합계 15
전체합계15
반복문이 종료됩니다
```

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - continue; 반복 구분 블록 한번만 건너고 나머지는 계속 수행

```
sum =0 ## → 초기식
for num in range(11) : ## → 시작값은 1, 끝값은 10,  증가값은 1
    sum = sum + num
    if num == 5 :
        print("5까지 합계" ,sum)
        continue
print("전체합계" ,sum)
print("반복문이 종료됩니다")
## →
5까지 합계 15
전체합계 55
반복문이 종료됩니다
```

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - 이중반복문(중첩반복문): 동일한 명령을 중첩하여 일정한 횟수만큼 실행하도록 만든 문장

초기식
for 제어변수 in range(조건식1: 시작값, 끝값+1, 증가값)
    조건식1을 만족하는 동안 실행할 문장 1(명령문, 함수)

    for 제어변수 in range(조건식2: 시작값, 끝값+1, 증가값)
        조건식2을 만족하는 동안 실행할 문장 2(명령문, 함수)

    조건식2 만족 조건이 종료되는 경우 실행할 문장 3(명령문, 함수)

조건식1 만족 조건이 종료되는 경우 실행할 문장 4(명령문, 함수)

반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - 이중반복문(중첩반복문): 동일한 명령을 중첩하여 일정한 횟수만큼 실행하도록 만든 문장

```
sum =0 ## → 초기식
for num in range(3) : ## → 시작값은 1, 끝값은 3,  증가값은 1
    sum = sum + num
    print("바깥쪽 반복문 합계 ",sum)
    for num2 in range(3) :  ## → 시작값은 1, 끝값은 3,  증가값은 1
        sum = sum + num2
        print( " 안쪽 반복문 합계 ",sum)
print("전체합계" ,sum)
print("반복문이 종료됩니다")
## →
```

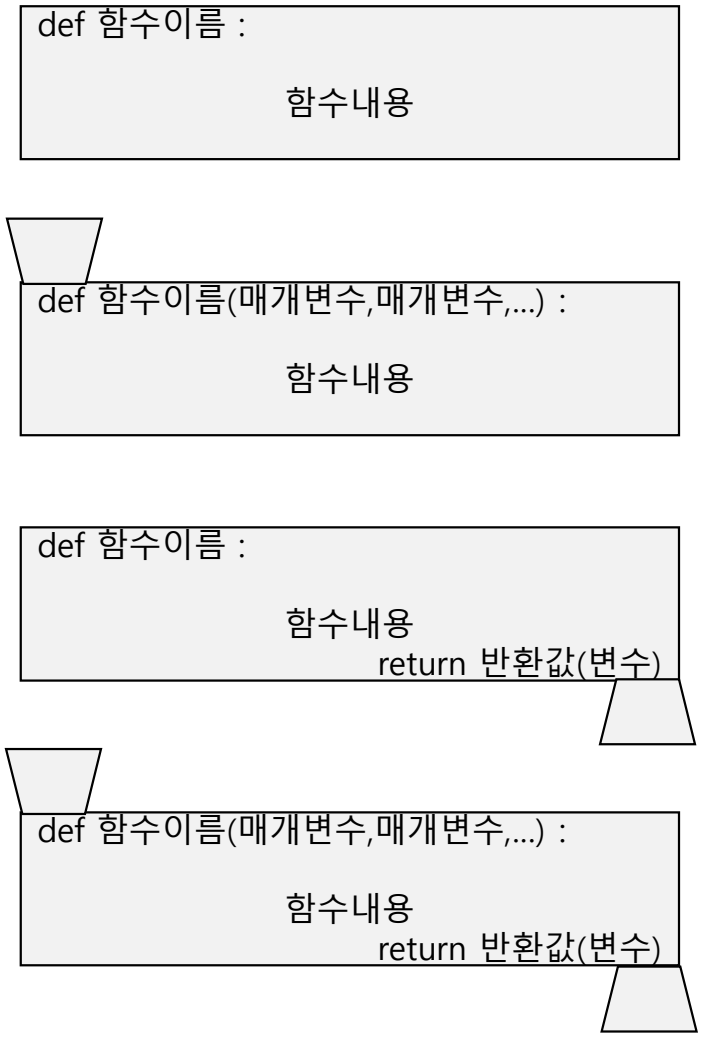
반복문

- 반복문: 동일한 명령을 계속 또는 일정한 횟수만큼 실행하도록 만든 문장
  - 이중반복문(중첩반복문): 동일한 명령을 중첩하여 일정한 횟수만큼 실행하도록 만든 문장

```
바깥쪽 반복문 합계  0
안쪽 반복문 합계  0
안쪽 반복문 합계  1
안쪽 반복문 합계  3
바깥쪽 반복문 합계  4
안쪽 반복문 합계  4
안쪽 반복문 합계  5
안쪽 반복문 합계  7
바깥쪽 반복문 합계  9
안쪽 반복문 합계  9
안쪽 반복문 합계  10
안쪽 반복문 합계  12
전체 반복문 합계  12
반복문이 종료됩니다
```

함수

- 함수형태
  - 기본형
    - 매개변수, 반환값(변수)이 없는 형태
  - 매개변수형
    - 매개변수만 있는 형태
  - 반환형
    - 반환값(변수)만 있는 형태, return
  - 매개변수, 반환형
    - 매개변수, 반환값(변수)가 있는 형태, return



함수

- 함수종류
  - 내장함수: 파이썬에서 제공하는(포함되어 있는) 함수
    - 함수호출, 매개변수, 반환값(변수) 사용
    - import 키워드 사용하지 않음
      - 문자열출력함수: print("출력") → 출력
      - 입력함수: mytext = input("문자를 입력하세요") → 안녕하세요
      - 절대값함수: a=abs(-10) → 10
      - 문자열실행함수: eval("1+2") → 3
      - 16진수함수: a = hex(3) → 0x3
      - 정수변환함수: a = int(3.4) → 3
      - 배열길이반환함수: a = len([1,2,3]) → 3
      - 최대값반환함수 a= max([1,2,3]) → 3
      - 최솟값반환함수 a= max([1,2,3]) → 1

함수

- 함수종류
  - 내장함수: 파이썬에서 제공하는(포함되어 있는) 함수
    - 함수호출, 매개변수, 반환값(변수) 사용
    - import 키워드 사용하지 않음
      - 파일함수: f = open(“mytest.txt”, “r”) → 파일
      - 리스트함수: list((1,2,3)) → [1,2,3]
      - 범위함수: list(rang(5)) → [0,1,2,3,4]
      - 제곱함수: pow(2,4) → 16
      - 반올림함수: round(4.6) → 5
      - 자료형반환함수: type(“abc”) → <class ‘str’>

함수

- 함수종류
  - 사용자정의함수: 사용자가 특정 작업을 수행하기 위해 만든 함수
    - 함수 이름 앞에 def 키워드를 사용
    - 함수 이름 다음 () 안에 매개변수 사용
    - () 다음에 : 붙임
    - 함수내용 작성
    - return 키워드 사용하여 반환값(변수) 지정

함수

- 함수종류
  - 사용자정의함수: 사용자가 특정 작업을 수행하기 위해 만든 함수
    - 재귀함수: 함수내용 안에 자기자신 함수(호출)를 사용한 함수

```
def MyHello(count):
    if count == 0:  # 종료 조건을 만듦. count가 0이면 다시 hello 함수를 호출하지 않고 끝냄
        return
    print("Hello World", count)
    count -= 1     # count를 1 감소시킨 뒤
    MyHello(count)  # 다시 Myhello에 넣음
MyHello(2)   # hello 함수 호출
```

→

```
Hello World, 2
Hello World, 1
```

함수

- 함수종류
  - 외장함수: 특별한 기능이 있는 모듈(함수의 집합)안에 있는 특정 함수를 사용
    - 함수를 포함시키기 위해 import 키워드를 사용
    - 시간모듈

```
import time
time.localtime() → time.struct_time(tm_year=2020, tm_mon=4, tm_mday=19, tm_hour=20, tm_min=20, tm_sec=40, tm_wday=6, tm_yday=110, tm_isdst=0):
# tm_wday는 요일(월요일~일요일, 0~6), tm_yday는 1월 1일부터 경과한 일수, tm_isdst는 서머타임 여부


import time
for i in range(10):
    print(i)
    time.sleep(2) → 2초 간격으로 0부터 9까지 숫자 출력
```

터틀 그래픽

- 터틀그래픽
  - 파이썬이 제공하는 클래스
  - 거북이의 모양을 가진 객체를 생성하여 선을 그려 디자인을 만들어 내는 것이 가능한 시각적인 도구
  - turtle 모듈 사용
    - import turtle

터틀 그래픽

- 터틀그래픽
  - 파이썬이 제공하는 클래스
  - 거북이의 모양을 가진 객체를 생성하여 선을 그려 디자인을 만들어 내는 것이 가능한 시각적인 도구
  - turtle 모듈 사용
    - import turtle

터틀 그래픽

- 터틀그래픽 함수
  t.forward() : 입력한 숫자만큼 앞으로 이동
  t.left() : 입력한 숫자만큼 왼쪽으로 각도 변경
  t.right() : 입력한 숫자만큼 오른쪽으로 각도 변경
  t.backward() : 입력한 숫자만큼 뒤로 이동
  t.goto(x,y) : 절대좌표 x,y로 이동

  t.penup() : 펜을 들기 (이동할 때 선을 그리지 않음)
  t.pendown() : 펜을 내리기 (이동할 때 선을 그림)

  t.isdown() : turtle의pen의 상태를 알려줌 (펜이 놓여져 있으면 True, 놓여있지 않으면 False)

터틀 그래픽

- 터틀그래픽 함수
  - t.circle(반지름길이) : 원점을 기준으로 지정한 반지름 길이에 해당하는 원을 그림
  - t.hideturtle() : turtle을 숨김
  - t.home() : turtle을 초기 위치로 되돌림
  - t.clear() : 화면을 초기 상태로 되돌림
  - t.setposition(x좌표,y좌표) : Turtle의 위치를 지정한 좌표(x,y)로 바꿔줌
  - t.position() : Turtle의 현재 위치를 나타냄

터틀 그래픽

- 터틀그래픽 함수
  t = turtle.Pen()
  t.bgcolor(): 배경색 바꾸기
  t.color(): 펜 색 바꾸기
  t.begin_fill() 펜 색 채우기 시작
  t.end_fill() 펜 색 채우기 종료

터틀 그래픽

- 터틀그래픽 함수

  t = turtle.Screen()

  t.setup(x,y) : 화면 크기 설정
  t.bgpic(“파일이름.gif") : 배경이미지 설정
  t.update() :배경 이미지 갱신

터틀 그래픽

- 터틀그래픽 함수

  t.setheading(숫자): 거북이가 바라보는 방향을 변경 (오른쪽 0도)
  t.clear() : 거북이 위치, 방향은 그대로 유지하고 환명을 지움
  t.speed(): 거북이 속도 조절 (1-10)
  t.shape(): 거북이 모양 변경 (arrow, turtle, circle, square, triangle, classic)
  t.fillcolor('색이름') : Turtle의색을지정한색으로변경 (white,red,green,blue)
  t.shapesize(w,h,b) : turtle의 크기를 원하는 세로w, 가로h,윤곽선b배로 변경

  t.done() : 터틀그래픽 종료

터틀 그래픽

- 터틀그래픽 사용
  - 객체의 방향을 정할때 기준과 방향을 정확하게 파악
  - setheading() 함수(메소드)를 사용하여 방향을 지정할때는 turtle의 헤드를 기준으로 왼쪽방향으로 각도를 계산
  - left()혹은 right() 함수 (메소드)를 사용할때는 turtle의 머리 기준으로 왼쪽인지 오른쪽인지 확인하고 각도를 지정
    - left(90) 또는 setheading(90) 동일
    - Right(90) 또는 setheading(270) 동일

튜플

- 튜플: 자료구조 형태중 하나로 순서가 있는 수정 불가능한 데이터의 집합
  - 접근, 삭제 가능, 추가 불가능
  - ( )괄호로 작성되어지며, 내부 원소는 ,로 구분
  - 튜플 이름 = (요소1, 요소2, 요소3,…)
  - a = (1,2,3,4,5)
    - >>> print(a) → (1,2,3,4,5)

튜플

- 튜플: 자료구조 형태중 하나로 순서가 있는 수정 불가능한 데이터의 집합
  - 0부터 시작하는 인덱스로 접근가능
    - >>> print(a[0]) → 1, print(a[1])→ 2, print(a[2])→ 3, print(a[4])→ 5, print(a[5])→ IndexError: tuple index out of range
  - 튜플 범위에 접근 콜론(시작:끝+1)
    - a = (1,2,3,4,5)
    - >>> print(a[1:3])
    - >>> print(a[1:2])
    - >>> print(a[1:1])
    - >>> print(a[0:1])
    - >>> print(a[2:])
    - >>> print(a[:4])
    - →
    - (2, 3)
    - (2,)
    - ( )
    - (1,)
    - (3, 4, 5)
    - (1, 2, 3, 4)

튜플

- 튜플: 자료구조 형태중 하나로 순서가 있는 수정 불가능한 데이터의 집합
  - 접근, 삭제 가능, 추가 불가능
  - ( )괄호로 작성되어지며, 내부 원소는 ,로 구분
  - 튜플 이름 = (요소1, 요소2, 요소3,…)
  - 튜플더하기 및 (정수)곱하기(반복) 연산 가능
  - a = (1,2,3,4,5)
    - >>> print(a) → (1,2,3,4,5)

튜플

- 튜플: 자료구조 형태중 하나로 순서가 있는 수정 불가능한 데이터의 집합
  - a = (1,2,3,4,5)
    - >>> print(a) → (1,2,3,4,5)
  - b=(10,20,30,"가","나")
    - >>> print(b) → (10,20,30,'가','나')
  - print(a+b)
  - print(a*3)
  - print(b*2)
  - print(a*b)
  - →
  - (1, 2, 3, 4, 5, 10, 20, 30, '가', '나')
  - (1, 2, 3, 4, 5, 1, 2, 3, 4, 5, 1, 2, 3, 4, 5)
  - (10, 20, 30, '가', '나', 10, 20, 30, '가', '나')
  - TypeError: can't multiply sequence by non-int of type 'tuple'

튜플

- 튜플: 자료구조 형태중 하나로 순서가 있는 수정 불가능한 데이터의 집합
  - a = (1,2,3,4,5,5)
  - len(a) → 튜플길이
  - max(a) → 최대요소
    - 튜플 안의 요소값들이 문자열과 숫자가 섞여 있을 경우 max() 메소드를 적용하면 TypeError
  - min(a) → 최소 요소
  - a.count(5): 요소 개수 → 요소 5가 2개
  - a.index(5): 요소 위치 인덱스 → 요소 5의 위치는 4
    - 똑같은 값이 2개 이상 들어있는 경우 처음 요소 값이 나타나는 위치의 index 를 반환
  - b = [1,2,3]
  - mytutple = tuple(b) → 리스트를 튜플로 변환

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - 접근, 추가, 삭제 가능
  - 중괄호({ })로 묶여 있으며 키와 값의 쌍으로 이루어지고 내부 원소는, 로 구분
  - 딕셔너리 이름= {키1:값1, 키2:값2, 키3:값3,…}
  - a = {1:”가”, 2:”나”, 3:”다”, 4:”라”, 5:”마”}
    - >>> print(a) → {1: '가', 2: '나', 3: '다', 4: '라', 5: '마'}
  - 딕셔너리 추가 → 딕셔너리 이름[키]=값
    - A[6]=“사”
    - >>> print(a) → {1: ' 가 ' , 2: ' 나 ' , 3: ' 다 ' , 4: ' 라 ' , 5: ' 마‘, 6:’사’}
  - 딕셔너리 추가하는 경우 이미 있는 키를 사용하면 쌍이 추가되는 것이 아닌 기존값 변경
    - a[6]=“상명”
    - >>> print(a) → {1: '가', 2: '나', 3: '다', 4: '라', 5: '마‘, 6:’상명’}

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - 접근, 추가, 삭제 가능
  - 딕셔너리 삭제 → del(딕셔너리 이름[키])
    - del(a[1])
    - >>> print(a) → {2: '나', 3: '다', 4: '라', 5: '마', 6:'상명'}
  - 딕셔너리 접근 → 딕셔너리이름.get(키) 함수
    - >>> print(a.get(6))
    - → 상명
  - 딕셔너리 모든 키 반환 → 딕셔너리이름.keys() 함수
    - >>> print(a.keys())
    - → dict_keys([2, 3, 4, 5, 6])
  - 딕셔너리 모든 값 반환 → 딕셔너리이름.values() 함수
    - >>> print(a.values())
    - → dict_values(['나', '다', '라', '마', '상명'])

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - 접근, 추가, 삭제 가능
  - 딕셔너리 모든 키를 리스트로 변환 → list(딕셔너리이름.keys())
    - >>> print(list(a.keys()))
    - → [2, 3, 4, 5, 6]
  - 딕셔너리 모든 값을 리스트로 변환 → list(딕셔너리이름.values())
    - >>> print(list(a.values()))
    - → [' 나 ' , ' 다 ' , ' 라 ' , ' 마 ' , ' 상명 ']
  - 딕셔너리를 튜플 형태로 변환 → 딕셔너리이름.items() 함수
    - >>> print(list(a.items()))
    - → dict_items([(2, '나'), (3, '다'), (4, '라'), (5, '마'), (6, '상명')])

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - 딕셔너리에 키가 있으면 True, 없으면 False 반환 → 키 in 딕셔너리이름
    - >>> print(a) → {2: '나', 3: '다', 4: '라', 5: '마', 6:'상명'}
    - >>> print(2 in a)
    - >>> print(20 in a)
    - →
    - True
    - False

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - for문 활용하여 딕셔너리 모든 값(key, value) 출력
    - >>> print(a) → {2: '나', 3: '다', 4: '라', 5: '마', 6:'상명'}
    - for k in a.keys() :
      - print("%i번째 딕셔러니 키는 %s 이고 값은 %s 입니다"%(k, k, a[k])
      - →
      - 2번째 딕셔러니 키는 2이고 값은 나 입니다
      - 3번째 딕셔러니 키는 3이고 값은 다 입니다
      - 4번째 딕셔러니 키는 4이고 값은 라 입니다
      - 5번째 딕셔러니 키는 5이고 값은 마 입니다
      - 6번째 딕셔러니 키는 6이고 값은 상명 입니다

딕셔너리

- 딕셔너리: 자료구조 형태중 하나로 쌍이 있는 수정 가능한 데이터의 집합
  - 접근, 추가, 삭제 가능
  - 딕셔너리 삭제 → del(딕셔너리 이름[키])
    - del(a[1])
    - >>> print(a) → {2: '나', 3: '다', 4: '라', 5: '마', 6:'상명'}
  - 딕셔너리 접근 → 딕셔너리이름.get(키) 함수
    - >>> print(a.get(6))
    - → 상명
  - 딕셔너리 모든 키 반환 → 딕셔너리이름.keys() 함수
    - >>> print(a.keys())
    - → dict_keys([2, 3, 4, 5, 6])
  - 딕셔너리 모든 값 반환 → 딕셔너리이름.values() 함수
    - >>> print(a.values())
    - → dict_values(['나', '다', '라', '마', '상명'])

집합

- 집합: 자료구조 형태 중 순서가 없는 수정 가능한 데이터의 집합
  - 접근, 연산 가능, 추가, 제거 가능, 중복허용 안됨
  - set 키워드를 이용, set()함수
  - set() 괄호 안에 리스트, 문자열 입력 가능
  - a = set(“상명대학교상명”)
    - >>> print(a) → {'교', '명', '학', '상', '대'}
  - 인덱스로 접근하려면 리스트 또는 튜플로 변환 (순서가 있는 데이터 집합으로 변환)

```
a = set(“상명대학교상명”)
print(a)
b = list(a)
print(b)
c = sorted(b)
print(c)
→
{'교', '명', '학', '상', '대'}
['교', '명', '학', '상', '대']
['교', '대', '명', '상', '학']
```

집합

- 집합: 자료구조 형태 중 순서가 없는 수정 가능한 데이터의 집합
  - 접근, 연산 가능, 추가, 제거 가능, 중복허용 안됨
  - set 키워드를 이용, set()함수
  - set() 괄호 안에 리스트, 문자열 입력 가능
  - a=set(1,2,3,4,5,1,2)
    - print(a) → TypeError: set expected at most 1 arguments, got 7
  - a=set([1,2,3,4,5,1,2])
    - print(a) → {1, 2, 3, 4, 5}
    - print(len(a)) → 5

집합

- 집합: 자료구조 형태 중 순서가 없는 수정 가능한 데이터의 집합
  - 집합 연산 → 합집합(OR,union), 교집합(AND,intersection), 차집합(-,difference)
  - a = set([1,2,3,4,5,1,2]) ## a={1,2,3,4,5,1,2}
  - b= set([2,4,6,8,10,2,4]) ## b={2,4,6,8,10,2,4}
    - print("집합a:",a) → 집합a: {1, 2, 3, 4, 5}
    - print("집합b:",b) → 집합b: {2, 4, 6, 8, 10}
    - print("합집합:",a|b) → 합집합: {1, 2, 3, 4, 5, 6, 8, 10}
    - print("합집합:",set.union(a,b)) → 합집합: {1, 2, 3, 4, 5, 6, 8, 10}
    - print("합집합:",a.union(b)) → 합집합: {1, 2, 3, 4, 5, 6, 8, 10}
    - print("교집합:",a&b) → 교집합: {2, 4}
    - print("교집합:",set.intersection(a,b)) →교집합: {2, 4}
    - print("교집합:",a.intersection(b)) →교집합: {2, 4}
    - print("차집합:",a-b) → 차집합: {1, 3, 5}
    - print("차집합:",set.difference(a,b)) →차집합: {1, 3, 5}
    - print("차집합:",a.difference(b)) →차집합: {1, 3, 5}

집합

- 집합: 자료구조 형태 중 순서가 없는 수정 가능한 데이터의 집합
  - 집합 연산 → 부분집합(subset), 진부분집합(proper subset)
  - 부분집합: 어떤 집합의 원소 중 일부만을 포함하는 집합
  - 진부분집합: 원소의 크기가 더 작은 부분집합
  - a =set([1,2,3,4,5,1,2]) ## a={1,2,3,4,5,1,2}
  - b=set([1,2,3,4,5]), ##b={1,2,3,4,5}
  - c=set([2,4]) ## c={(2,4}
    - print("부분집합",a.issubset(b)) → 부분집합 True
    - print("부분집합",b.issubset (a)) → 부분집합 True
    - print("부분집합",c.issubset(a)) → 부분집합 Ture
    - print("진부분집합",a.issuperset(b)) → 진부분집합 True
    - print("진부분집합",b.issuperset(a)) → 진부분집합 True
    - print("진부분집합",c.issuperset(a)) → 진부분집합 False

집합

- 집합: 자료구조 형태 중 순서가 없는 수정 가능한 데이터의 집합
  - 집합 연산 → 집합의 크기(cardinality)
  - a =set([1,2,3,4,5,1,2]) ## a={1,2,3,4,5,1,2}
  - b=set([1,2,3,4,5]), ##b={1,2,3,4,5}
  - c=set([2,4]) ## c={(2,4}
    - print(len(a)) → 5
    - print(len(b)) → 5
    - print(len(c)) → 2

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력: 키보드로 입력, 모니터로 출력
    - 표준 입력: input() → 키보드, 문자열
    - 표준 출력: print() → 모니터, 문자열

  - 파일 입출력: 파일로 입력(읽음), 파일로 출력(씀)
    - 파일 입력(읽음)
      - read()
      - readline()
      - readlines()
    - 파일 출력(씀)
      - write()
      - writelines()

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 표준 입력: input() → 표준 출력: print()
    - 표준 입력: input() → 파일 출력: write(), writellness()
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - 파일 입력: read(), readline(), readlines() → 파일 출력: write(), writelines()

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 파일열기
    - open() 함수에서 파일명을 지정, 읽기인지 쓰기인지를 지정
    - open() 함수의 마지막 매개변수를 모드
      - 변수이름 = open("파일이름", "r") → 파일읽기(입력)
      - 변수이름 = open("파일이름", "w") → 파일쓰기(출력)

      - 파일열기모드
        - r: 읽기모드 기본값
        - w: 쓰기모드 기본에 파일이 있으면 덮어씀
        - r+: 읽기/쓰기 겸용 모드
        - a: 쓰기모드, 기존에 파일이 있으면 이어서 씀 (Append)
        - t: 택스트모드, 텍스트 파일을 처리, 기본값
        - b: 바이너리 모드, 바이너리 파일(이진파일) 처리

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 파일처리
    - 파일에 데이터 쓰거나 파일로부터 데이터를 읽어올 수 있는 상태
  - 파일닫기
    - 변수이름.close()

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
      - 메모장에 텍스트 입력:

        컴퓨터 프로그램 어떤 문제를 해결하기 위해 컴퓨터에게 주어지는 처리 방법과 순서를 기술한 일련의 명령문의 집합

        컴퓨터 프로그래밍 컴퓨터가 이해할 수 있는 규칙에 따라 프로그램 수행절차를 프로그래밍 언어로 작성하는 것

        프로그래밍 알고리즘은 프로그래밍 언어를 사용하여 어떠한 문제를 해결하기 위한 명령어 모임
      - 메모장 텍스트 저장: 인코딩 UTF-8로 저장
      - 메모장 파일 이름: data.txt

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - read() → 일정 길이 만큼 텍스트를 읽음

```
inFile = None
inString =“”
inFile = open(“C:/test/data.txt”, “r”, encoding=“utf-8”)
inString = inFile.read(2) # 값-1
print(inString, end=“”)
inFile.close()
→
컴
```

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - readline() → 텍스트를 한 줄씩 읽음

```
inFile = None
inString =“”
inFile = open(“C:/test/data.txt”, “r”, encoding=“utf-8”)
inString = inFile.readline()
print(inString, end=“”)
inString = inFile.readline()
print(inString, end=“”)
inString = inFile.readline()
print(inString, end=“”)
inString = inFile.readline()
print(inString, end=“”)
inFile.close()
```

→

컴퓨터 프로그램 어떤 문제를 해결하기 위해 컴퓨터에게 주어지는 처리 방법과 순서를 기술한 일련의 명령문 집합
컴퓨터 프로그래밍 컴퓨터가 이해할 수 있는 규칙에 따라 프로그램 수행절차를 프로그래밍 언어로 작성하는 것
프로그래밍 알고리즘은 프로그래밍 언어를 사용하여 어떠한 문제를 해결하기 위한 명령어 모임

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - readline() → 텍스트를 한 줄씩 읽음

```
inFile = None
inString =“”
inFile = open(“C:/test/data.txt”, “r”, encoding=“utf-8”)
while True :
        inString = inFile.readline()
        if inString==“”:
                break;
        print(inString, end=“”)
inFile.close()
```

→

컴퓨터 프로그램 어떤 문제를 해결하기 위해 컴퓨터에게 주어지는 처리 방법과 순서를 기술한 일련의 명령문 집합

컴퓨터 프로그래밍 컴퓨터가 이해할 수 있는 규칙에 따라 프로그램 수행절차를 프로그래밍 언어로 작성하는 것

프로그래밍 알고리즘은 프로그래밍 언어를 사용하여 어떠한 문제를 해결하기 위한 명령어 모임

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - readlines() → 텍스트 모든 줄을 한번에 읽음

```
inFile = None
inList = []
inString = “”
inFile = open(“C:/test/data.txt”, “r”, encoding=“utf-8”)
inList = inFile.readlines()
for inString in inList:
        print(inString, end=“”)
inFile.close()
```

→

컴퓨터 프로그램 어떤 문제를 해결하기 위해 컴퓨터에게 주어지는 처리 방법과 순서를 기술한 일련의 명령문 집합
컴퓨터 프로그래밍 컴퓨터가 이해할 수 있는 규칙에 따라 프로그램 수행절차를 프로그래밍 언어로 작성하는 것
프로그래밍 알고리즘은 프로그래밍 언어를 사용하여 어떠한 문제를 해결하기 위한 명령어 모임

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 파일 입력: read(), readline(), readlines() → 표준 출력: print()
    - readlines() → 텍스트 모든 줄을 한번에 읽음

```
import os
def file_read_test(inFile):
    inList, inString = [], ""
    if os.path.exists(inFile) :
        inFile = open(inFile, "r", encoding="utf-8")
        inList = inFile.readlines()
        for inString in inList:
            print(inString, end="")
        inFile.close()
    else :
        print(“%s 파일이 없습니다”%(inFile)

def test():
    inFile = input("파일 이름을 입력하세요")
    file_read_test(inFile)

test()
 →
파일 이름을 입력하세요C:/test/data.txt
```

파일입출력

- 파일입출력: 파일 입력과 출력 과정에 필요한 함수
  - 표준 입출력과 파일 입출력
    - 표준 입력: input() → 파일 출력: write(), writellness()

```
outFile = None
outString = ""
outFile = open(" C:/test/data2.txt", "w", encoding="utf-8")
while True :
    outString = input("메모를 입력하세요:")
    if outString !="" :
        outFile.writelines(outString +"\n")
    else :
        break
outFile.close()
```

→

```
메모를 입력하세요:안녕하세요
메모를 입력하세요:
```

→

data2.txt 생성여부 확인

프로그래밍 문법

- 파이썬: 파이썬은 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체 지향적, 동적 타이핑 대화형 언어

  - 고급 프로그래밍 언어
    - 고급프로그래밍언어: 사람이 이해하기 쉽게 작성된 프로그래밍 언어 (C++, Java, Python), 가독성
      컴파일러, 인터프리터에 의해 저급 프로그래밍언어로 번역되어 실행
    - 저급프로그래밍언어: 기계(컴퓨터)가 이해하기 쉽게 작성된 프로그래밍 언어(기계어, 어셈블리어)

  - 인터프리터식
    - 인터프리터식언어: 코드를 컴파일(Compile)하지 않고도 바로 실행할 수 있는 프로그래밍 언어
    - 컴파일 언어: 코드를 (한꺼번에) 기계어로 번역되어 실행할 수 있는 프로그래밍 언어

프로그래밍 문법

- 파이썬: 파이썬은 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체 지향적, 동적 타이핑 대화형 언어
  - 객체 지향적 언어
    - 객체지향적 언어: 물리적, 추상적인 것 중에서 자신의 속성을 가지고 있고 다른 것과 식별 가능한 것 (데이터와 함수)을 프로그래밍 언어로 표현 가능 (C++, Java, Python)
    - 절차지향적: 수행되어야 할 연속적인 계산(수행) 과정을 포함하여 프로그래밍 언어로 표현 가능 (C)
  - 대화형 언어
    - Command Prompt 환경 제공: Python 3.6.5 Shell (파이썬 명령프롬프트)
    - GUI(Graphical User Interface) 환경 제공: Turtle Graphic, Tkinter

프로그래밍 문법

- 파이썬: 파이썬은 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체 지향적, 동적 타이핑 대화형 언어

  - 파이썬 객체: 클래스
    - 클래스 정의와 생성

      class 클래스 이름:

        #클래스 데이터1,… → 필드1,…

        #클래스 함수1,…→ 메소드1

프로그래밍 문법

- 파이썬: 파이썬은 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체 지향적, 동적 타이핑 대화형 언어

프로그래밍 문법

- 파이썬: 파이썬은 고급 프로그래밍 언어로, 플랫폼에 독립적이며 인터프리터식, 객체지향적, 동적 타이핑 대화형 언어

  - 파이썬 객체: 클래스
    - 클래스 정의와 생성

```
class Car:
    #필드
    car_number=“”
    car_color=“”
    car_speed=0
    #메소드
    def speed_up(pedal_value):
        self.car_speed = self.car_speed + pedal_value
        print(“자동차 속도는 %d 입니다”%(self.car_speed))
    def check_car_color(color):
        self.car_color = color
        print(“자동차는 %색 입니다”%(self.car_color))
```

프로그래밍 문법

- 클래스 정의와 생성과 인스턴스

```
class Car:
        #필드
        car_number=“”
        car_color=“”
        car_speed=0
        #메소드
        def speed_up(pedal_value):
                self.car_speed = self.car_speed + pedal_value
                print(“자동차 속도는 %d 입니다”%(self.car_speed))
        def check_car_color(color):
                self.car_color = color
                print(“자동차는 %색 입니다”%(self.car_color))
myCar_first = Car()
myCar_first.numer=“1234”
myCar_first.speed_up(10) → 자동차 속도는 10입니다
myCar_second = Car()
myCar_second.number”4567”
myCar_second.speed_up(20) → 자동차 속도는 20입니다
```

프로그래밍 문법

- 파이썬 객체: 클래스와 인스턴스, 생성자
  - 생성자: 인스턴스를 생성하면 호출되는 메소드 (초기화 과정 작성)

```
class 클래스 이름:
    def __init__(self,매개변수,…): ##클래스 생성자 (초기화)
        self.클래스 데이터변수 이름= 매개변수
        self.클래스 데이터변수 이름…

    def 함수이름(self,매개변수,…):
        print(“self.클래스 데이터변수 이름”)

myinstance1 = 클래스이름(데이터1,…)
myinstance2 = 클래스이름(데이터2,…)
```

프로그래밍 문법

- 파이썬 객체: 클래스와 인스턴스
  - 필드나 메소드 사용

```
class 클래스 이름:
    def __init__(self,매개변수,…):
        self.클래스 데이터변수 이름= 매개변수
        self.클래스 데이터변수 이름…

    def 함수이름(self,매개변수,…):
        print(“self.클래스 데이터변수 이름”)

myinstance1 = 클래스이름(데이터1,…)
myinstance2 = 클래스이름(데이터2,…)

myinstance1.필드이름1 = 값1
myinstance1.필드이름 2= 값2
myinstance1.메소드이름1()
myinstance1.메소드이름2()
```

프로그래밍 문법

- 클래스 정의와 생성과 인스턴스, 생성자

```
class Car:
        def __init__(self,va11, val2,val3):
                self.car_number = val1
                self.car_color=val2
                self.car_speed=val3

        def speed_up(self,pedal_value):
                        self.car_speed = self.car_speed + pedal_value
                        print("자동차 속도는 %d 입니다"%(self.car_speed))
        def check_car_color(self, color):
                        self.car_color = color
                        print("자동차는 %색 입니다"%(self.car_color))
myCar_first = Car("1234","Black",10)
myCar_first.speed_up(10) → 자동차 속도는 20입니다
myCar_second = Car("4567","Silver",20)
myCar_second.speed_up(20) → 자동차 속도는 40입니다
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
  - Button: 간단한 버튼
  - Canvas: 화면에 어떤 것을 그릴 때 사용
  - Checkbutton: 2가지의 구별되는 값을 가지는 변수를 표현
  - Entry: 한 줄의 텍스를 입력받는 필드
  - Label: 텍스트나 이미지를 표시
  - Menu: 메뉴를 표시(풀다운, 팝업 메뉴)
  - Radiobutton: 여러 값을 가질 수 있는 변수를 표시
  - Text: 형식을 가지는 텍스트를 표시(여러 가지 스타일과 속성으로 텍스트를 표시)

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
  - 버튼이 있는 윈도우

```
from tkinter import *
window = Tk()
button = Button(window, text="버튼입니다")
button.pack()
window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
  - 라벨과 엔트리(텍스트박스)가 있는 윈도우

```
from tkinter import *
window = Tk()

Label1 = Label(window, text="첫번째")
Label2 = Label(window, text="두번째")
Label1.pack()
Label2.pack()



Entry1 = Entry(window)
Entry2 = Entry(window)
Entry1.pack()
Entry2.pack()

window.mainloop()
```

프로그래밍 문법

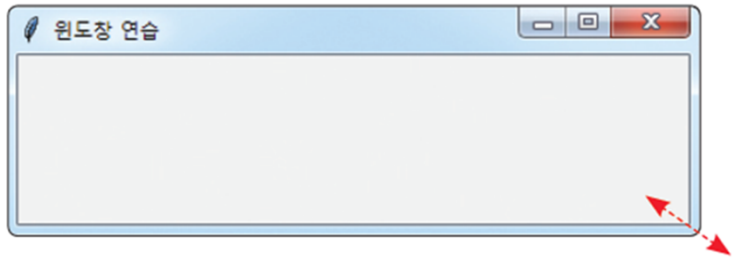
- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
  - 위치 배치

```
Label1 = Label(window, text="라벨1", bg="red", fg="white")
Label2 = Label(window, text="라벨2", bg="green", fg="white")
Label3 = Label(window, text="라벨1", bg="blue", fg="white")

Label1.place(x=0,y=0)
Label2.place(x=20,y=20)
Label3.place(x=40,y=40)

window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
- 윈도우 창의 기본 구성
  - 위젯(Widget) - 윈도우 창에 나올 수 있는 문자, 버튼, 체크박스, 라디오버튼 등

```
from tkinter import *
##tkinter는 파이썬에서 GUI 관련 모듈을 제공해주는 표준 윈도우 라이브러리

window = Tk()
## Tk( )는 기본이 되는 윈도우를 반환, 루트 윈도우(Root Window)

window.title("윈도우 창 연습")
## 윈도우 창에 제목을 표시

window.geometry("400 x 100")
## 윈도우 창의 초기 크기를 400×100으로 지정

window.resizable(width=FALSE, height = FALSE)
## 가로와 세로의 크기가 변경되지 않도록 설정

window.mainloop()
```



프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈

```
from tkinter import *
window = Tk()

label1 = Label(window, text="상명대학교")
label2 = Label(window, text="융합공과대학", font=("맑은 고딕",30), fg="blue")
label3 = Label(window, text="화이팅", fg="green", width=20, height=5, anchor=SE)
## anchor는 위젯이 어느 위치에 자리 잡을지를 지정함, anchor에 사용할 수 있는 값은
N, NE, E, SE, S, SW, W, NW, CENTER 등이며 기본값은 CENTER임

label1.pack() ## pack() 함수를 통해 해당 라벨을 화면에 표시
label2.pack()
label3.pack()

window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈

```
from tkinter import *
window = Tk()

photo = PhotoImage(file="gif/test.gif")
label1 = Label(window, image = photo)

label1.pack()

window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
- 버튼
  - 마우스로 클릭하면 눌리는 효과와 함께 지정한 작업이 실행되게 하는 위젯

```
from tkinter import *
window = Tk()

button1 =Button(window, text="상명대학교", fg="red",, command = quit)

button1.pack()

window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈

```
from tkinter import *
def myFunc():
    messagebox.showinfo("버튼테스트","버튼이 눌렸습니다 " )

window =Tk()

photo = PhotoImage(file="gif/test.gif")
button1 =Button(window, image=photo, command=myFunc)

button1.pack()

window.mainloop()
```

프로그래밍 문법

- Tkinter: 파이썬에서 그래픽 사용자 인터페이스(GUI: graphical user interface)를 개발할 때 필요한 모듈
- 체크버튼
  - 켜기/끄기를 위해 사용되는 위젯, 형식은 'Checkbutton(부모 윈도우, 옵션…)'
  - 체크버튼을 켜거나 끄면 메시지 창이 나오게 하는 프로그램

```
from tkinter import *
def myFunc():
    if chk.get() == 0 :
        messagebox.showinfo("","체크버튼이 설정되었습니다 " )
    else :
        messagebox.showinfo("","체크버튼이 해제되었습니다 " )
chk = IntVar() ##IntVar() 함수는 정수형 형식의 변수를 생성

cb1.pack()

window.mainloop()
```

프로그래밍 문법

- 라디오버튼
  - 여러 개 중 하나를 선택하는 데 사용되는 위젯. 형식은 'Radiobutton(부모 윈도우, 옵션…)'

```
from tkinter import *
def myFunc():
        if var.get() == 1 :
                label1.configure(text="1학년")
        elif var.get() == 2 :
                label1.configure(text="2학년")
        elif var.get() == 3 :
                label1.configure(text="3학년")
        elif var.get() == 4 :
                label1.configure(text="4학년")
var = IntVar() ##IntVar() 함수는 정수형 형식의 변수를 생성
rb1 = Radiobutton(window, text="1학년", variable=var, value=1, command=myFunc)
rb2 = Radiobutton(window, text="1학년", variable=var, value=1, command=myFunc)
rb3 = Radiobutton(window, text="1학년", variable=var, value=1, command=myFunc)
rb4 = Radiobutton(window, text="1학년", variable=var, value=1, command=myFunc)
label1 = Label(window, text="선택한 학년", fg="red")
rb1.pack()
rb2.pack()
rb3.pack()
rb4.pack()
label1.pack()
window.mainloop()
```

프로그래밍 문법

- 수평으로 정렬
  - 파이썬에서 위젯을 출력할 때는 pack( )이나 place( ) 함수를 사용하며, 기존에 출력된 위젯의 모양을 변경할 때는 configure( ) 함수를 사용
  - pack( )함수의 옵션 중 수평으로 정렬하는 방법으로 'side =LEFT,RIGHT' 방식

```
from tkinter import *
window = Tk()
button1 = Button(window, text="버튼')
button2 = Button(window, text="버튼')
button3 = Button(window, text="버튼')

button1.pack(side=LEFT)
button2.pack(side=LEFT)
button3.pack(side=LEFT)

window.mainloop()
```

프로그래밍 문법

- 수평으로 정렬

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for i in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = RIGHT)

window.mainloop()
```

프로그래밍 문법

- 수직으로 정렬

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for i in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = TOP)
    ##btn.pack(side = BOTTOM)

window.mainloop()
```

프로그래밍 문법

- 위젯 폭 맞춰 정렬

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for i in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = TOP, fill=X)

window.mainloop()
```

프로그래밍 문법

- 위젯 사이에 여백 주기

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for i in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = TOP, fill=X, padx = 10, pady=10)

window.mainloop()
```

프로그래밍 문법

- 위젯 사이에 내부 여백 주기

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for i in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = TOP, fill=X, ipadx = 10, ipady=10)

window.mainloop()
```

프로그래밍 문법

- 위젯 사이에 내외부 여백 주기

```
from tkinter import *
window = Tk()
btnList = [None] * 3
for I in range(0,3):
    btnList[i] = Button(window, text="버튼"+str(i+1))

for btn in btnList:
    btn.pack(side = TOP, fill=X, ipadx = 10, ipady=10, padx = 10, pady=10)

window.mainloop()
```

프로그래밍 문법

- 마우스 이벤트
  - 키보드 및 마우스를 누르는 것을 이벤트(Event)
  - mainloop( ) 함수는 이벤트가 발생하기를 대기하는 함수

```
def 이벤트처리함수(event):
    # 이 영역에 마우스이벤트 발생 시
        작동할 내용 코딩

위젯.bind("마우스이벤트", 이벤트처리함수)
```

마우스 작동	관련 마우스 버튼	이벤트 코드
클릭할 때	모든 버튼 공통	〈Button〉
	왼쪽 버튼	〈Button-1〉
	가운데 버튼	〈Button-2〉
	오른쪽 버튼	〈Button-3〉
떼었을 때	모든 버튼 공통	〈ButtonRelease〉
	왼쪽 버튼	〈ButtonRelease-1〉
	가운데 버튼	〈ButtonRelease-2〉
	오른쪽 버튼	〈ButtonRelease-3〉
더블클릭할 때	모든 버튼 공통	〈Double-Button〉
	왼쪽 버튼	〈Double-Button-1〉
	가운데 버튼	〈Double-Button-2〉
	오른쪽 버튼	〈Double-Button-3〉
드래그할 때	왼쪽 버튼	〈B1-Motion〉
	가운데 버튼	〈B2-Motion〉
	오른쪽 버튼	〈B3-Motion〉
마우스 커서가 위젯 위로 올라왔을 때		〈Enter〉
마우스 커서가 위젯에서 떠났을 때		〈Leave〉

프로그래밍 문법

- 마우스 이벤트
  - 키보드 및 마우스를 누르는 것을 이벤트(Event)
  - mainloop( ) 함수는 이벤트가 발생하기를 대기하는 함수

```
def clickLeft(event):
    messagebox.showinfo("마우스",
    "마우스왼쪽 버튼 클릭")

window = Tk()
window.bind("<button-1>", clickLeft)
window.mainloop()
```

마우스 작동	관련 마우스 버튼	이벤트 코드
클릭할 때	모든 버튼 공통	〈Button〉
	왼쪽 버튼	〈Button-1〉
	가운데 버튼	〈Button-2〉
	오른쪽 버튼	〈Button-3〉
떼었을 때	모든 버튼 공통	〈ButtonRelease〉
	왼쪽 버튼	〈ButtonRelease-1〉
	가운데 버튼	〈ButtonRelease-2〉
	오른쪽 버튼	〈ButtonRelease-3〉
더블클릭할 때	모든 버튼 공통	〈Double-Button〉
	왼쪽 버튼	〈Double-Button-1〉
	가운데 버튼	〈Double-Button-2〉
	오른쪽 버튼	〈Double-Button-3〉
드래그할 때	왼쪽 버튼	〈B1-Motion〉
	가운데 버튼	〈B2-Motion〉
	오른쪽 버튼	〈B3-Motion〉
마우스 커서가 위젯 위로 올라왔을 때		〈Enter〉
마우스 커서가 위젯에서 떠났을 때		〈Leave〉

프로그래밍 문법

- 마우스 이벤트
  - 키보드 및 마우스를 누르는 것을 이벤트(Event)
  - mainloop( ) 함수는 이벤트가 발생하기를 대기하는 함수

```
def clickImage(event):
    messagebox.showinfo("마우스",
   "이미지 클릭")

window = Tk()
window.geometry("400x400")
photo = PhotoImage(file="gif/test.gif")
label1 = Label(window, image=photo)
label1.bind("<Button>",clickImage)
label1.pack(expand=1, anchor=CENTER)
window.mainloop()
```

마우스 작동	관련 마우스 버튼	이벤트 코드
클릭할 때	모든 버튼 공통	〈Button〉
	왼쪽 버튼	〈Button-1〉
	가운데 버튼	〈Button-2〉
	오른쪽 버튼	〈Button-3〉
떼었을 때	모든 버튼 공통	〈ButtonRelease〉
	왼쪽 버튼	〈ButtonRelease-1〉
	가운데 버튼	〈ButtonRelease-2〉
	오른쪽 버튼	〈ButtonRelease-3〉
더블클릭할 때	모든 버튼 공통	〈Double-Button〉
	왼쪽 버튼	〈Double-Button-1〉
	가운데 버튼	〈Double-Button-2〉
	오른쪽 버튼	〈Double-Button-3〉
드래그할 때	왼쪽 버튼	〈B1-Motion〉
	가운데 버튼	〈B2-Motion〉
	오른쪽 버튼	〈B3-Motion〉
마우스 커서가 위젯 위로 올라왔을 때		〈Enter〉
마우스 커서가 위젯에서 떠났을 때		〈Leave〉

프로그래밍 구현

맺음말