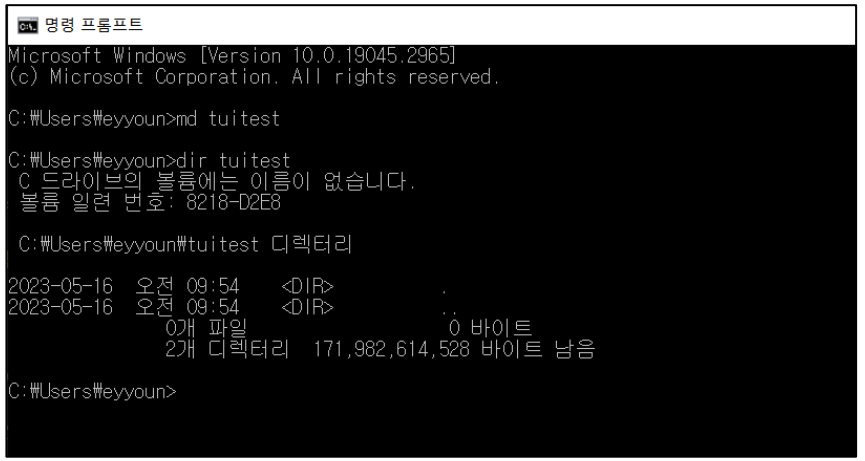
GUI Programming

TUI vs GUI

- TUI (Text-based User Interface)와 GUI (Graphical User Interface)는 사용자와 컴퓨터 간의 상호작용을 위한 두 가지 주요한 인터페이스 유형
- TUI (Text-based User Interface):
  - TUI는 텍스트 기반으로 구성된 사용자 인터페이스
  - 일반적으로 명령줄 인터페이스 또는 콘솔 인터페이스로 알려져 있음
  - TUI에서는 사용자는 명령어를 입력하고 텍스트 기반으로 정보를 표시하며, 키보드를 사용하여 상호작용
  - 주로 터미널 또는 커맨드 프롬프트에서 사용
  - TUI의 예로는 명령어 기반의 MS DOS, 리눅스 셸, 텍스트 편집기



TUI vs GUI

- TUI의 장점:
  - 자원 소모가 적고 경량
  - 텍스트 기반이므로 콘솔 환경에서도 사용 가능
  - 간단하고 빠르게 작성할 수 있음
  - 스크립트와의 통합이 용이
- TUI의 단점:
  - 사용자가 명령어나 옵션을 기억하고 입력해야 함
  - 그래픽 요소가 제한적이므로 복잡한 시각적 표현이 어려움

TUI vs GUI

- GUI (Graphical User Interface):
  - GUI는 그래픽 요소를 사용하여 사용자와 상호작용하는 인터페이스
  - 사용자는 윈도우, 버튼, 메뉴, 입력 필드 등과 같은 그래픽 요소를 마우스로 조작하여 작업을 수행할 수 있음
  - 대부분의 현대적인 응용 프로그램은 GUI를 사용하며, 대표적인 예로 웹 브라우저, 워드 프로세서, 그래픽 편집 도구 등
- GUI의 장점:
  - 시각적인 요소를 통해 사용자에게 직관적인 환경을 제공
  - 복잡한 작업을 그래픽 요소를 통해 간편하게 수행할 수 있음
  - 다양한 입력 방식을 지원하므로 사용자의 편의성을 높일 수 있음
- GUI의 단점:
  - 자원 소모가 많고 용량이 큰 경우가 있음
  - 구현 및 디자인에 시간과 비용이 많이 들 수 있음
  - 다양한 플랫폼에서 동일한 모습을 유지하기 어려울 수 있음

Python GUI Programming

- 파이썬은 다양한 GUI 프로그래밍 도구와 라이브러리를 제공하여 GUI 애플리케이션 개발을 지원
- 다양한 선택지:
  - 파이썬은 다양한 GUI 도구와 라이브러리를 제공
  - Tkinter, PyQt, PySide, Kivy, wxPython 등의 도구와 라이브러리를 사용할 수 있음
  - 사용자가 프로젝트 요구사항과 개발 선호도에 맞는 가장 적합한 도구를 선택
- 쉬운 학습 환경:
  - Tkinter는 파이썬의 표준 GUI 도구로 간단하고 쉽게 사용
  - 파이썬 자체에 포함되어 있어 별도의 설치가 필요하지 않으며, 문법과 구조가 상대적으로 간단
  - 파이썬을 처음 접하는 개발자도 비교적 쉽게 GUI 애플리케이션을 개발
- 크로스 플랫폼 지원:
  - 파이썬은 크로스 플랫폼 개발을 지원하므로 동일한 코드를 사용하여 여러 플랫폼에서 GUI 애플리케이션을 실행
  - Tkinter와 같은 도구는 Windows, macOS, Linux와 같은 다양한 운영체제에서 작동

Python GUI Programming

- 확장성:
  - 파이썬은 다른 언어와의 통합이 용이하며, C/C++로 작성된 라이브러리와의 결합을 통해 추가적인 기능을 제공
  - 예를 들어, PyQt나 wxPython은 Qt 또는 wxWidgets라는 C++ 기반의 GUI 프레임워크와 결합하여 파이썬에서 사용할 수 있는 다양한 기능을 제공
- 풍부한 기능과 위젯:
  - 파이썬 GUI 도구는 다양한 기능과 위젯을 제공하여 사용자 인터페이스를 다양하게 구성
  - 버튼, 레이블, 텍스트 상자, 체크박스, 라디오 버튼, 드롭다운 메뉴 등 다양한 위젯을 활용하여 사용자와 상호작용하는 요소를 구현
  - 그래픽 요소를 처리하는 기능도 제공되어 도표, 차트, 이미지 표시 등과 같은 고급 기능을 개발
- 데이터 시각화:
  - 파이썬은 데이터 시각화를 위한 다양한 라이브러리인 Matplotlib, Seaborn, Plotly 등을 제공
  - 파이썬 GUI에서 데이터를 시각적으로 표현하고 그래프, 플롯, 차트 등을 생성할 수 있음
  - 데이터 시각화는 데이터 분석 및 보고에 필수적인 요소로 사용

Python GUI - Tkinter

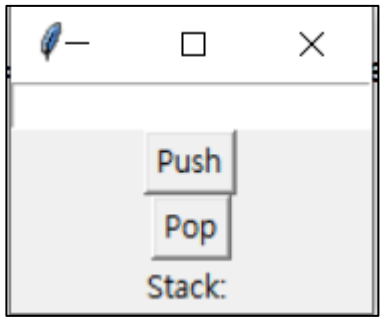
- Tkinter는 파이썬의 표준 GUI 패키지로서 다양한 위젯을 제공
  - 위젯(widget)은 사용자 인터페이스를 구성하는 요소
  - 위젯은 버튼, 레이블, 입력 필드, 체크 박스, 라디오 버튼 등 다양한 형태의 사용자 인터랙션 요소를 포함
  - 위젯들은 사용자와 애플리케이션 간의 상호작용을 담당하며, 애플리케이션의 기능을 제어하고 정보를 표시하는 역할

- Label (레이블): 사용자에게 텍스트나 이미지를 표시하는 데 사용
- Button (버튼): 사용자가 클릭할 수 있는 버튼을 생성
- Entry (입력 필드): 사용자로부터 텍스트를 입력받을 수 있는 상자
- Checkbutton (체크 박스): 사용자가 체크할 수 있는 옵션을 표시, 여러 개의 체크 박스를 그룹으로 묶어 다중 선택 가능

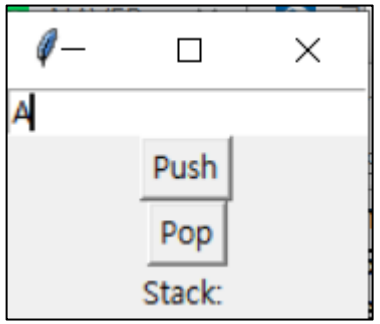
Python GUI - Tkinter

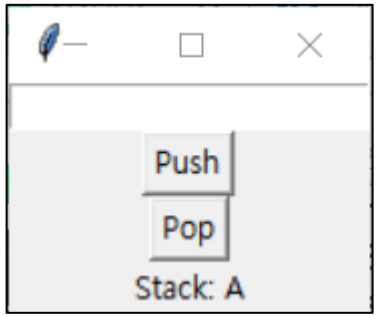
- Radiobutton (라디오 버튼): 여러 개의 옵션 중 하나를 선택할 수 있는 버튼, 여러 개의 라디오 버튼을 그룹으로 묶어 단일 선택 가능
- Listbox (리스트 박스): 여러 항목을 리스트 형태로 표시하고 사용자가 선택할 수 있도록 하며, 단일 선택과 다중 선택이 가능
- Combobox (콤보 박스): 텍스트 상자와 드롭다운 목록을 결합한 위젯, 사용자가 선택할 수 있는 목록을 제공하며, 자유롭게 텍스트 입력 가능
- Text (텍스트 상자): 여러 줄의 텍스트를 입력받거나 표시하는 데 사용, 대용량 텍스트의 편집이 가능하며 스크롤바 포함 가능

Example(StackGUI)

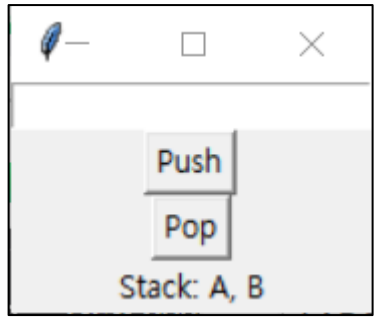


Push : A

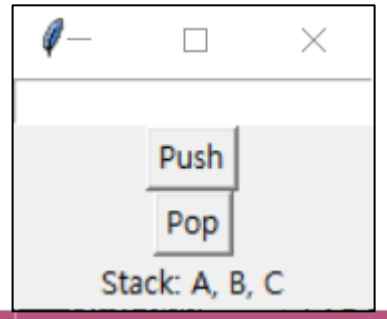




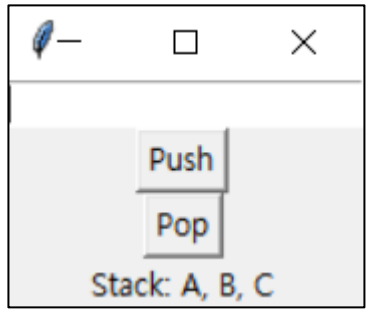
Push : B



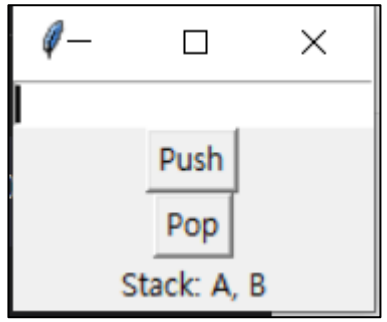
Push : C



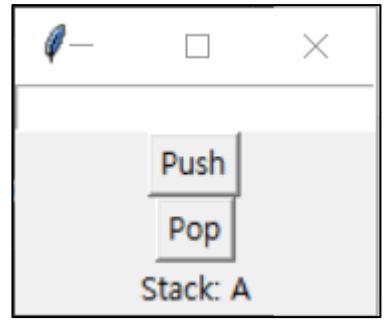
Pop



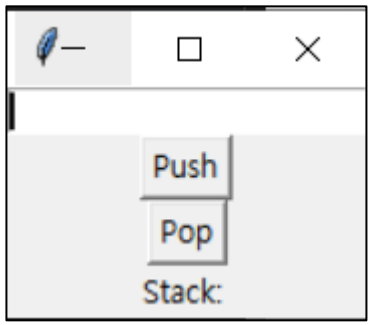
Pop



Pop



Pop



Example(StackGUI)

```
import tkinter as tk

class StackGUI:
    def __init__(self):
        self.stack = []  # 스택을 저장할 리스트

        self.window = tk.Tk()  # Tkinter 창 생성
        self.window.title("Stack Example")  # 창 제목 설정

        # 입력창과 버튼 생성
        self.input_entry = tk.Entry(self.window)
        self.input_entry.pack()

        self.push_button = tk.Button(self.window, text="Push", command=self.push)
        self.push_button.pack()

        self.pop_button = tk.Button(self.window, text="Pop", command=self.pop)
        self.pop_button.pack()

        # 스택 출력을 위한 레이블 생성
        self.stack_label = tk.Label(self.window, text="Stack: ")
        self.stack_label.pack()
```

Example(StackGUI)

```
    def push(self):
        item = self.input_entry.get()  # 입력창에서 값을 가져옴
        if item:
            self.stack.append(item)  # 스택에 값 추가
            self.update_stack_label()  # 스택 레이블 업데이트
            self.input_entry.delete(0, tk.END)  # 입력창 초기화

    def pop(self):
        if self.stack:
            self.stack.pop()  # 스택에서 값 제거
            self.update_stack_label()  # 스택 레이블 업데이트

    def update_stack_label(self):
        stack_text = "Stack: " + ", ".join(self.stack)  # 스택 내용을 문자열로 변환
        self.stack_label.config(text=stack_text)  # 레이블 업데이트

    def run(self):
        self.window.mainloop()  # 창 실행

# StackGUI 객체 생성 및 실행
stack_gui = StackGUI()
stack_gui.run()
```

Example(StackGUI)

- Tkinter를 사용하여 스택 예제 구현
- 사용자는 데이터를 입력하여 스택에 push하고 pop 버튼을 누를 때마다 최종 스택 상태가 보여짐

- self.input_entry: Tkinter Entry 위젯으로 사용자가 데이터를 입력할 수 있는 입력창
- self.push_button: Tkinter Button 위젯으로 push 함수를 호출하는 버튼
- self.pop_button: Tkinter Button 위젯으로 pop 함수를 호출하는 버튼
- self.stack_label: Tkinter Label 위젯으로 현재 스택 상태를 표시하는 레이블
- push : 입력창에서 값을 가져와 스택에 추가하고, 스택 레이블을 업데이트
- pop : 스택에서 값을 제거하고, 스택 레이블을 업데이트
- update_stack_label : 스택 리스트를 문자열로 변환하여 스택 레이블의 텍스트를 업데이트
- run : Tkinter의 이벤트 루프를 실행하여 사용자의 상호작용을 처리하고 GUI를 업데이트

Example(StackGUI)

- 예제에서 사용된 주요 Tkinter 함수

- tk.Tk(): Tkinter의 Tk 클래스를 이용하여 주 창을 생성, Tkinter 애플리케이션의 루트 윈도우를 초기화
- self.window.title(): 생성된 창의 제목을 설정, title() 메서드를 사용하여 창의 제목을 지정
- tk.Entry(): Tkinter의 Entry 클래스를 이용하여 텍스트 입력 필드를 생성, Entry 위젯은 사용자가 텍스트를 입력할 수 있는 입력창을 제공
- self.input_entry.get(): Entry 위젯에서 입력된 값을 가져옴. get() 메서드를 호출하여 현재 입력된 텍스트를 가져올 수 있음
- tk.Button(): Tkinter의 Button 클래스를 이용하여 버튼을 생성, Button 위젯은 사용자가 클릭할 수 있는 버튼을 제공하며 버튼을 클릭하면 지정된 함수 또는 메서드가 호출

Example(StackGUI)

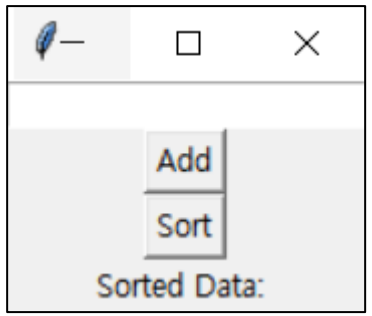
- 예제에서 사용된 주요 Tkinter 함수

- self.push_button.pack(): Button 위젯을 화면에 배치, pack() 메서드를 호출하여 위젯을 현재 윈도우에 자동으로 배치
- self.pop_button.pack(): Button 위젯을 화면에 배치, pack() 메서드를 호출하여 위젯을 현재 윈도우에 자동으로 배치
- self.stack_label.config(): Label 위젯의 속성을 변경, config() 메서드를 사용하여 레이블의 속성을 수정할 수 있음 예제에서는 텍스트(text) 속성을 수정하여 스택의 내용을 업데이트
- self.window.mainloop(): Tkinter 애플리케이션의 이벤트 루프를 실행, mainloop() 메서드는 Tkinter 애플리케이션의 이벤트를 처리하고 창을 업데이트하는 데 사용됨 애플리케이션의 실행을 지속시키기 위해 이 메서드를 호출해야 함

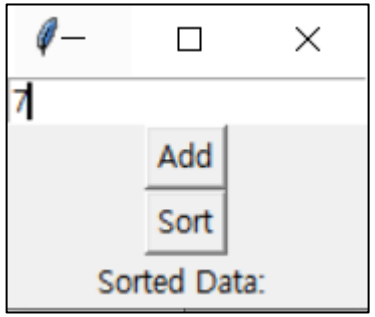
Example(BubbleSortGUI)

- Tkinter를 사용하여 버블 소트 예제를 구현
- 사용자는 입력창에 숫자를 입력한 후 "Add" 버튼을 클릭하여 숫자를 리스트에 추가
- "Sort" 버튼을 클릭하면 입력된 숫자를 버블 소트 알고리즘을 사용하여 정렬
- 정렬된 결과는 "Sorted Data" 레이블에 표시

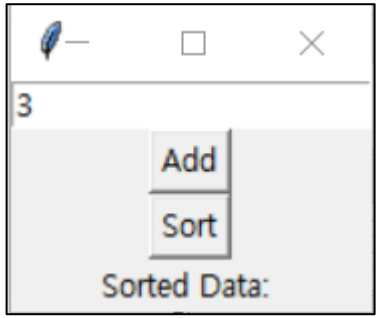
Example(BubbleSortGUI)



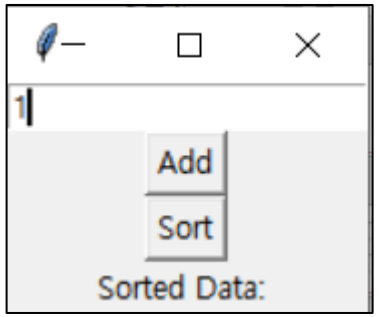
Add : 7



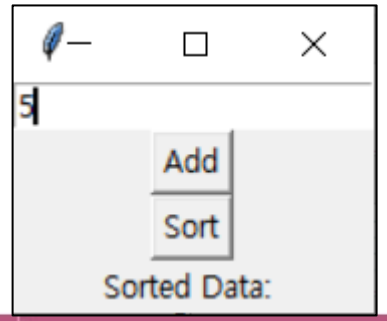
Add : 3

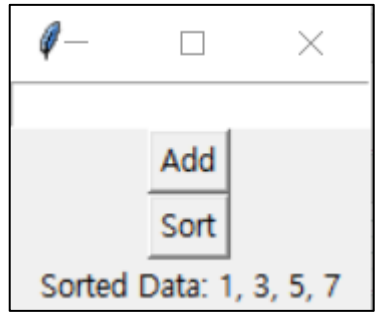


Add : 1



Add : 5





Example(BubbleSortGUI)

```
import tkinter as tk

class BubbleSortGUI:
    def __init__(self):
        self.data = []  # 정렬할 데이터를 저장할 리스트
        self.sorted_data = []  # 정렬된 데이터를 저장할 리스트

        self.window = tk.Tk()  # Tkinter 창 생성
        self.window.title("Bubble Sort Example")  # 창 제목 설정

        # 입력창과 버튼 생성
        self.input_entry = tk.Entry(self.window)
        self.input_entry.pack()

        self.add_button = tk.Button(self.window, text="Add", command=self.add_data)
        self.add_button.pack()

        self.sort_button = tk.Button(self.window, text="Sort", command=self.bubble_sort)
        self.sort_button.pack()

        # 정렬된 데이터를 표시할 레이블 생성
        self.sorted_label = tk.Label(self.window, text="Sorted Data: ")
        self.sorted_label.pack()
```

Example(BubbleSortGUI)

```
    def add_data(self):
        value = self.input_entry.get()  # 입력창에서 값을 가져옴
        if value:
            self.data.append(int(value))  # 입력된 값을 정수로 변환하여 리스트에 추가
            self.input_entry.delete(0, tk.END)  # 입력창 초기화

    def bubble_sort(self):
        n = len(self.data)
        self.sorted_data = self.data.copy()  # 정렬할 데이터를 복사하여 정렬된 데이터 리스트 초기화

        for i in range(n - 1):
            for j in range(n - i - 1):
                if self.sorted_data[j] > self.sorted_data[j + 1]:
                    self.sorted_data[j], self.sorted_data[j + 1] = self.sorted_data[j + 1], self.sorted_data[j]
                    self.update_sorted_label()  # 정렬된 데이터 레이블 업데이트

    def update_sorted_label(self):
        sorted_text = "Sorted Data: " + ", ".join(map(str, self.sorted_data))  # 정렬된 데이터를 문자열로 변환
        self.sorted_label.config(text=sorted_text)  # 레이블 업데이트

    def run(self):
        self.window.mainloop()  # 창 실행

# BubbleSortGUI 객체 생성 및 실행
bubble_sort_gui = BubbleSortGUI()
bubble_sort_gui.run()
```

Example(BubbleSortGUI)

- BubbleSortGUI 클래스 기능:
- __init__(self): 클래스의 초기화 메서드, data와 sorted_data 리스트를 초기화하고 Tkinter 창을 생성하고 버튼, 레이블을 초기화
- add_data(self): 사용자가 입력한 값을 가져와서 data 리스트에 추가, 입력창의 값을 초기화
- bubble_sort(self): data 리스트의 값들을 버블 소트 알고리즘을 사용하여 정렬, 정렬된 결과는 sorted_data 리스트에 저장되며, 정렬 과정에서 update_sorted_label() 메서드를 호출하여 정렬된 데이터를 표시하는 레이블을 업데이트
- update_sorted_label(self): 정렬된 데이터를 문자열로 변환한 후 sorted_label 레이블의 텍스트를 업데이트
- run(self): Tkinter 창을 실행

Example(BubbleSortGUI)

- tk.Tk(): Tkinter의 Tk 클래스를 이용하여 주 창을 생성, Tkinter 애플리케이션의 루트 윈도우를 초기화
- self.window.title(): 생성된 창의 제목을 설정, title() 메서드를 사용하여 창의 제목을 지정
- tk.Entry(): Tkinter의 Entry 클래스를 이용하여 텍스트 입력 필드를 생성, Entry 위젯은 사용자가 텍스트를 입력할 수 있는 입력창을 제공
- self.input_entry.pack(): 입력창을 화면에 배치, pack() 메서드를 사용하여 위젯을 창에 레이아웃
- tk.Button(): Tkinter의 Button 클래스를 이용하여 버튼을 생성, 버튼은 사용자가 클릭할 수 있는 인터페이스 요소, 버튼을 생성할 때 text 매개변수로 버튼에 표시될 텍스트를 지정하고, command 매개변수로 버튼이 클릭되었을 때 실행될 함수를 지정
- self.sort_button.pack(): 정렬 버튼을 화면에 배치
- self.sorted_label = tk.Label(self.window, text="Sorted Data: "): 레이블을 생성, 레이블은 텍스트를 표시하는 역할, 생성할 때 text 매개변수로 초기 텍스트를 지정
- self.sorted_label.pack(): 정렬된 데이터를 표시하는 레이블을 화면에 배치

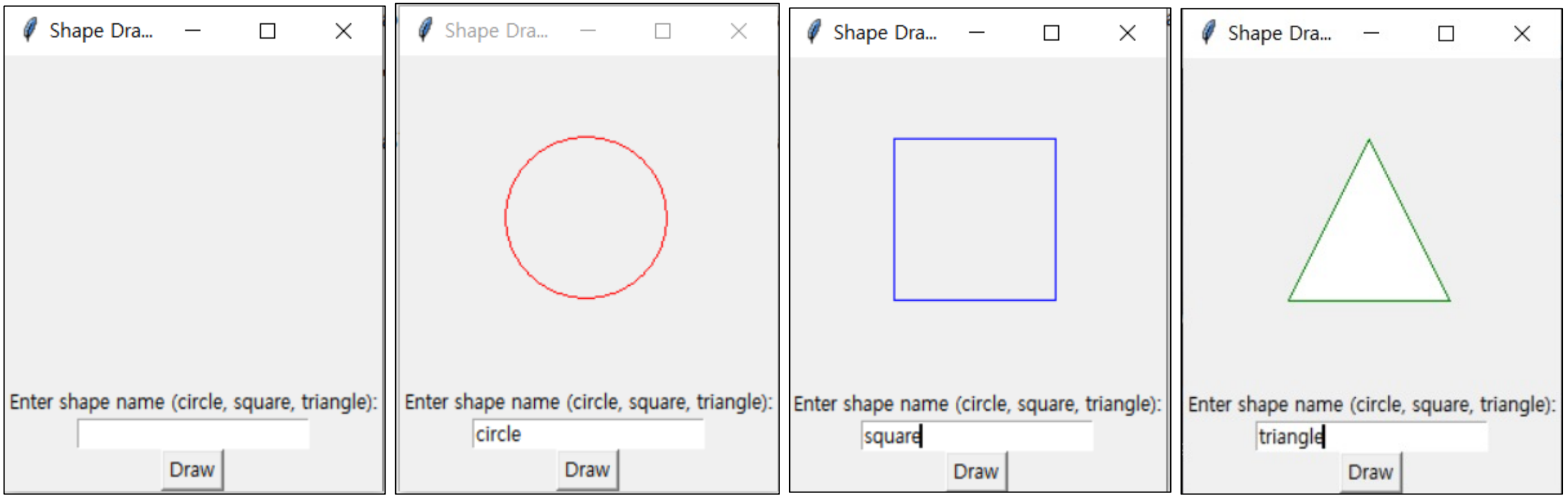
Example(BubbleSortGUI)

- self.input_entry.get(): 입력창의 값을 가져오며, get() 메서드를 사용하여 입력된 값을 얻을 수 있음
- self.data.append(int(value)): 입력된 값을 정수로 변환하여 self.data 리스트에 추가
- self.input_entry.delete(0, tk.END): 입력창의 내용을 삭제하여 초기화, delete() 메서드의 첫 번째 매개변수로 시작 인덱스를, 두 번째 매개변수로 종료 인덱스를 지정하여 삭제할 범위를 지정할 수 있음 tk.END는 입력창의 끝을 의미
- self.bubble_sort(): 버블 소트 알고리즘을 호출하여 데이터를 정렬
- self.sorted_data[j], self.sorted_data[j + 1] = self.sorted_data[j + 1], self.sorted_data[j]: 두 개의 값을 스왑하여 정렬하는 과정을 수행, 버블 소트 알고리즘에서 인접한 두 값의 비교와 교환을 나타냄
- self.update_sorted_label(): 정렬된 데이터 레이블을 업데이트
- self.sorted_label.config(text=sorted_text): 레이블의 텍스트를 업데이트, config() 메서드를 사용하여 레이블의 속성을 변경할 수 있음
- self.window.mainloop(): Tkinter 창을 실행하고 이벤트 루프를 시작, 이 메서드를 호출하면 창이 열

Example(Shape)

- Tkinter를 사용하여 도형 그리기 예제 구현
- 사용자로부터 도형 이름을 입력받아 원, 네모, 세모 도형 그리기
- Tkinter를 사용하여 GUI를 구성하고, 클래스, 상속, 메소드 오버라이딩을 활용하여 각 도형의 동작을 정의

Example(Shape)



Example(Shape)

```
import tkinter as tk

class Shape:
    def __init__(self, shape_name):
        self.shape_name = shape_name

    def draw(self):
        pass

class Circle(Shape):
    def __init__(self, shape_name):
        super().__init__(shape_name)

    def draw(self):
        canvas.create_oval(50, 50, 150, 150, outline='red')

class Square(Shape):
    def __init__(self, shape_name):
        super().__init__(shape_name)

    def draw(self):
        canvas.create_rectangle(50, 50, 150, 150, outline='blue')

class Triangle(Shape):
    def __init__(self, shape_name):
        super().__init__(shape_name)

    def draw(self):
        canvas.create_polygon(50, 150, 100, 50, 150, 150, outline='green', fill='white')
```

Example(Shape)

```
def draw_shape():
    shape_name = entry.get()

    if shape_name.lower() == 'circle':
        shape = Circle(shape_name)
    elif shape_name.lower() == 'square':
        shape = Square(shape_name)
    elif shape_name.lower() == 'triangle':
        shape = Triangle(shape_name)
    else:
        shape = None

    if shape:
        shape.draw()
    else:
        print("Invalid shape name.")
```

Example(Shape)

```
window = tk.Tk()
window.title("Shape Drawing")

canvas = tk.Canvas(window, width=200, height=200)
canvas.pack()

label = tk.Label(window, text="Enter shape name (circle, square, triangle):")
label.pack()

entry = tk.Entry(window)
entry.pack()

button = tk.Button(window, text="Draw", command=draw_shape)
button.pack()

window.mainloop()
```

Example(Shape)

- Shape 클래스를 정의하여 도형의 부모 클래스로 활용
- Circle, Square, Triangle 클래스는 Shape 클래스를 상속받아 draw() 메소드를 오버라이딩
- 각각의 클래스는 해당 도형을 그리는 메소드를 정의하며, canvas 객체를 사용하여 그리기 작업을 수행

- draw_shape() 함수는 사용자가 입력한 도형 이름을 확인하고 해당하는 도형 객체를 생성한 후 draw() 메소드를 호출하여 도형을 그림
- 윈도우 창을 생성하고 위젯들을 배치한 후 window.mainloop()를 호출하여 이벤트 루프를 실행하여 윈도우 창이 표시되고 사용자 입력을 기다림

Example(Shape)

- 예제에 사용된 Tkinter의 Canvas 메소드
- create_oval(x1, y1, x2, y2, options)
  - 원을 그리는 메소드
  - (x1, y1)과 (x2, y2)는 원의 왼쪽 위와 오른쪽 아래 좌표
  - options 매개변수는 원의 속성을 설정하는데 사용
  - 예를 들어 outline='red'와 같이 외곽선 색상을 지정
- create_rectangle(x1, y1, x2, y2, options)
  - 사각형을 그리는 메소드
  - (x1, y1)과 (x2, y2)는 사각형의 대각선 꼭지점 좌표
  - options 매개변수는 사각형의 속성을 설정하는데 사용
  - 예를 들어 outline='blue'와 같이 외곽선 색상을 지정

Example(Shape)

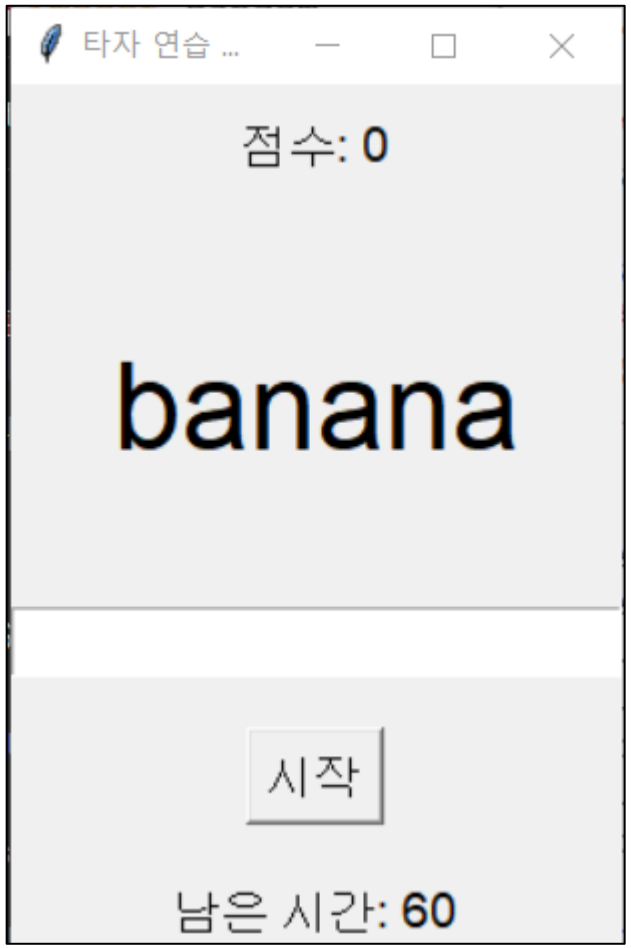
- 다각형을 그리는 메소드
  - (x1, y1), (x2, y2), ...는 다각형의 꼭지점 좌표
  - options 매개변수는 다각형의 속성을 설정하는데 사용
  - 예를 들어 outline='green'와 같이 외곽선 색상을 지정

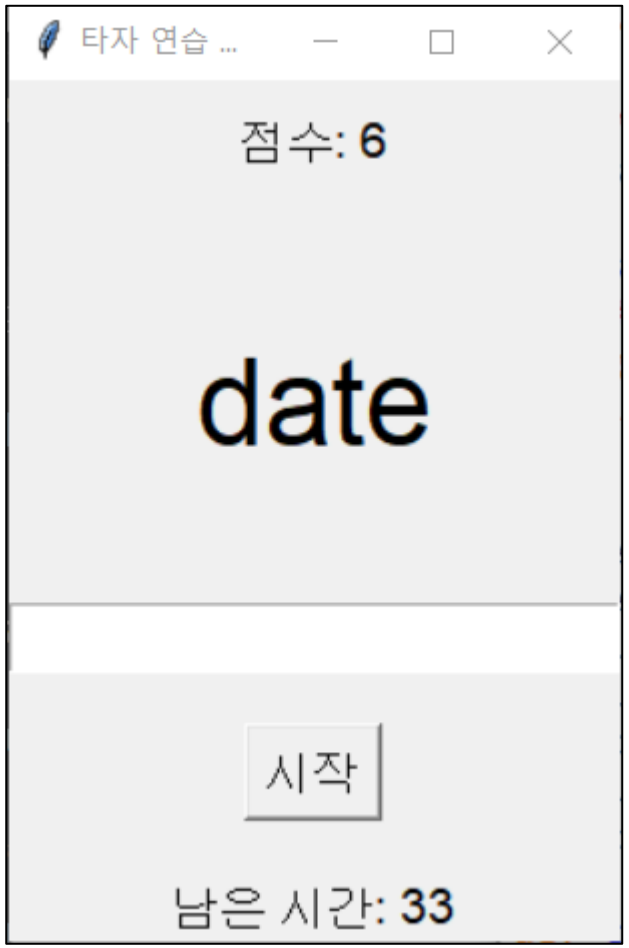
- 이 세 가지 메소드는 Canvas 객체에 그림을 그리는 데 사용
- Canvas은 Tkinter에서 그래픽을 그리는 공간을 나타내며, 위의 메소드들을 호출하여 도형을 생성하고 배치

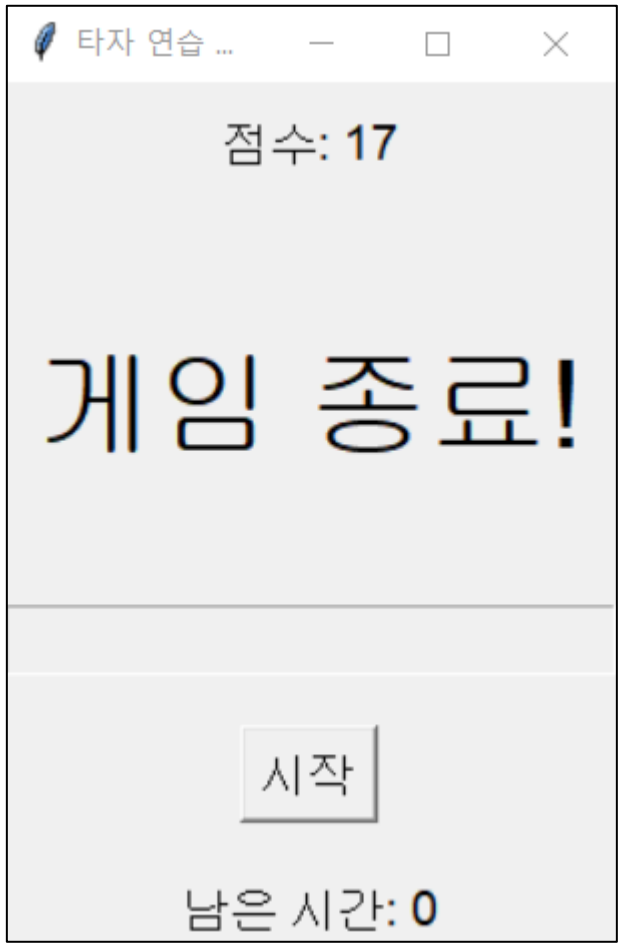
- 각 메소드는 도형의 위치와 크기를 좌표로 지정하고, options 매개변수를 사용하여 도형의 속성을 설정
- 속성은 다양한 옵션으로 지정할 수 있으며, 예를 들어 outline, fill, width 등이 있음
- outline은 도형의 외곽선 색상, fill은 도형의 내부 색상, width는 선의 두께 등을 설정

Example(TypingGame)

- Tkinter를 사용하여 타자 연습 예제 구현







Example(TypingGame)

```
import tkinter as tk
import random
import time

# 단어 목록
words = ['apple', 'banana', 'cherry', 'date', 'elderberry', 'fig', 'grape']

def start_game():
    # 게임 시작 함수
    global score, time_left, current_word
    if time_left == 60:  # 처음 게임 시작 시 초기화
        score = 0
        time_left = 60
    entry.focus_set()  # 입력 상자에 포커스 설정
    entry.delete(0, tk.END)  # 입력 상자 초기화
    label_score.config(text="점수: {}".format(score))
    label_time_left.config(text="남은 시간: {}".format(time_left))
    current_word = random.choice(words)  # 랜덤한 단어 선택
    label_word.config(text=current_word)
    countdown()
```

Example(TypingGame)

```
def check_word():
    # 입력한 단어 확인 함수
    global score, current_word
    input_word = entry.get().strip().lower()
    if input_word == current_word:
        score += 1
    entry.delete(0, tk.END)
    label_score.config(text="점수: {}".format(score))
    current_word = random.choice(words)
    label_word.config(text=current_word)

def countdown():
    # 시간 카운트다운 함수
    global time_left
    if time_left > 0:
        time_left -= 1
        label_time_left.config(text="남은 시간: {}".format(time_left))
        window.after(1000, countdown)
    else:
        label_word.config(text="게임 종료!")
        entry.delete(0, tk.END)
        entry.config(state=tk.DISABLED)
```

```python
# Tkinter 윈도우 생성
window = tk.Tk()
window.title("타자 연습 게임")

# 점수 표시 레이블
score = 0
label_score = tk.Label(window, text="점수: {}".format(score), font=("Helvetica", 14))
label_score.pack(pady=10)

# 단어 표시 레이블
current_word = random.choice(words)
label_word = tk.Label(window, text=current_word, font=("Helvetica", 36))
label_word.pack(pady=50)

# 입력 상자
entry = tk.Entry(window, font=("Helvetica", 16))
entry.pack()

# Enter 키 입력 시 단어 확인
entry.bind("<Return>", lambda event: check_word())

# 시작 버튼
start_button = tk.Button(window, text="시작", font=("Helvetica", 14), command=start_game)
start_button.pack(pady=20)

# 남은 시간 표시 레이블
time_left = 60
label_time_left = tk.Label(window, text="남은 시간: {}".format(time_left), font=("Helvetica", 14))
label_time_left.pack()

window.mainloop()
```

Example(TypingGame)

- 게임의 목표는 제한 시간 내에 표시된 단어를 입력하고 엔터 키를 눌러 맞추는 것
- 맞춘 단어의 개수에 따라 점수가 증가하며, 게임이 끝나면 최종 점수가 표시

- `start_game()`: 게임 시작 함수로, 게임을 초기화하고 시간을 카운트다운
- `check_word()`: 입력한 단어를 확인하고 정답인 경우 점수를 증가
- `countdown()`: 남은 시간을 카운트다운하여 게임을 제한 시간 내에 진행
- `entry.bind("<Return>", lambda event: check_word())`: 엔터 키 입력 시 `check_word()` 함수를 호출하여 단어를 확인
- 나머지 부분은 윈도우 생성, 레이블 표시, 입력 상자 및 시작 버튼 설정 등과 관련된 `Tkinter` 코드

Example(TypingGame)

- Typing Game 코드에서 사용된 Tkinter 메서드

- tk.Tk(): Tkinter의 Tk 클래스를 사용하여 윈도우를 생성, Tkinter 애플리케이션의 루트 윈도우
- window.title("타자 연습 게임"): title() 메서드를 사용하여 윈도우의 제목을 설정
- tk.Label(): 텍스트나 이미지를 표시하는 레이블 위젯을 생성, Label 클래스의 인스턴스를 생성하고, 부모 윈도우와 함께 사용될 텍스트, 폰트 등을 지정
- label.pack(): pack() 메서드를 사용하여 레이블 위젯을 윈도우에 배치, 이 메서드는 위젯을 윈도우에 자동으로 배치
- tk.Entry(): 텍스트를 입력할 수 있는 입력 상자 위젯을 생성

Example(TypingGame)

- Typing Game 코드에서 사용된 Tkinter 메서드
- entry.bind("<Return>", lambda event: check_word()): bind() 메서드를 사용하여 입력 상자에서 <Return> (엔터) 키를 누를 때 check_word() 함수를 호출, 이를 통해 입력 상자에서 엔터 키를 누를 때마다 단어 확인과 관련된 기능이 동작
- tk.Button(): 버튼 위젯을 생성, Button 클래스의 인스턴스를 생성하고, 부모 윈도우와 함께 사용될 텍스트, 폰트 및 버튼을 클릭했을 때 실행될 콜백 함수를 지정
- button.pack(): 버튼 위젯을 윈도우에 배치
- window.mainloop(): Tkinter 애플리케이션의 이벤트 루프를 실행, 이 메서드를 호출하여 윈도우를 표시하고 사용자의 동작을 대기하며, 사용자의 입력 및 이벤트에 따라 적절한 동작을 수행할 수 있음

- 코드에서 사용된 Tkinter 메서드는 윈도우, 레이블, 버튼, 입력 상자 등 다양한 위젯을 생성하고 배치하는 데 사용되며, 이를 통해 타자 연습 게임의 GUI를 구성하고 사용자와 상호작용할 수 있는 인터페이스를 제공