GUI（tkinter）を題材にしたモジュールの利活用

はじめに

これまでもいろいろなアプリを作成してきましたが，いずれも入力・出力は文字列，つまりテキストベースのプログラムを作成してきました．ここからは，より実践的な内容として，tkinterというPythonの標準ライブラリを使用してGUI（グラフィカルユーザインターフェイス）アプリケーションを作成する方法を学びます．

tkinterとは

tkinterは，Pythonに標準で付属しているGUIツールキットです．これを使用することで，ボタン，テキストフィールド，キャンバスなどの様々なウィジェット＝部品を持つウィンドウアプリケーションを作成することができます．tkinterでは，ウィンドウアプリを構成する部品の設計図としてクラスが定義され，我々はそれのオブジェクトを生成して利用します．様々な用途に応じてクラスの概念を使って整えられたPythonのパッケージ活用の具体例の一つとして，試してみましょう．

基本的な仕組み

tkinterでは、一般的に以下の手順でアプリケーションを作成します：

1. tkinterモジュールをインポートする
2. メインウィンドウを作成する
3. ウィジェット（ボタン，ラベル，テキストフィールドなど）を作成し，配置する．あわせてイベント処理も記述する
4. イベントループを開始する

次の例は，最小限のウィンドウを表示するアプリです．単一のソースコードとしてでも，ノートブックのセルに書き込んでも実行可能です．

```
import tkinter as tk

# メインウィンドウの作成
root = tk.Tk()
# ウィンドウのタイトルを設定（オプション）
root.title("最小限のアプリ")
# メインループの開始
root.mainloop()
```

まず，import文でtkinterモジュールをインポートしこれから使えるようにします．このとき，「as tk」は，これからtkinterに含まれる関数やクラス等の属性は，「tk.オブジェクト名」として表現できるように定義することを表します． 2行目で，実際にtk.Tk()としてtkinterの中心となるウィンドウクラスのインスタンスを生成してrootという変数に割り当てます．このウィンドウは，アプリケーションの基本となって，ウィジェットと呼ぶ様々な部品を配置するためのコンテナとして機能します． 3行目では，ウィンドウクラスのメソッドであるtitleでタイトルを設定しています． 4行目は，ウィンドウを表示し，ユーザーの操作（マウスクリックやキーボード入力など）に反応するためのイベント待機状態に入ります．このループは，ウィンドウが閉じられるまで継続し，アプリケーションの実行を維持します．一般的なGUIプログラムの最後に呼び出される重要な部分です．

簡単な例：カウンターアプリ

このようなまっさらなウィンドウに，tkinterで定義されるいろいろなGUI部品を載せ，さらにその動作を定義することで，アプリ全体を構築します．

まずは，へぇーボタンを押すとカウントが増えるシンプルなアプリケーションを例に見てみましょう．

```python
import tkinter as tk

#スーパークラスがない場合は(object)とつけなくてもOK
class CounterApp:
    def __init__(self, master):
        # masterはtkinterのルートウィンドウ（Tkインスタンス）を参照
        self.master = master

        # ウィンドウのタイトルを設定
        self.master.title("カウンターアプリ")
        # ウィンドウサイズを設定
        self.master.geometry("300x200")

        # カウンターの初期値を0に設定
        self.count = 0

        # カウントを表示するラベルを作成
        self.label = tk.Label(self.master, text="へぇー: 0")
        # ラベルをウィンドウに配置
        self.label.pack()

        # カウントアップボタンを作成。クリック時にincrementメソッドを呼び出す
        self.button = tk.Button(self.master, text="へぇー", command=self.increment, fg="re
        # ボタンをウィンドウに配置
        self.button.pack()

    def increment(self):
        # カウントを1増やす
        self.count += 1
        # ラベルのテキストを更新して新しいカウントを表示
        self.label.config(text=f"へぇー: {self.count}")

root = tk.Tk()
app = CounterApp(root)
root.mainloop()
```

このコードを実行すると、以下のような機能を持つウィンドウが表示されます：

- 現在のカウントを表示するラベル
- クリックするとカウントが1増えるボタン

アプリとしてのクラスの定義

まずプログラムの構成に注目しましょう．この例ではGUIアプリケーションをクラスベースで構築する方法を示しています．tk.Tk()でメインウィンドウを作成するのは最初の例と同じですが，次に上で定義したCounterAppクラスのインスタンスを，メインウィンドウのオブジェクトであるrootを与えつつ生成しています．つまり，カウンターアプリとしての機能を持った一塊としてクラスをまず定義し，それをtkinerのお決まりの一連の手続きの中で利用しています．

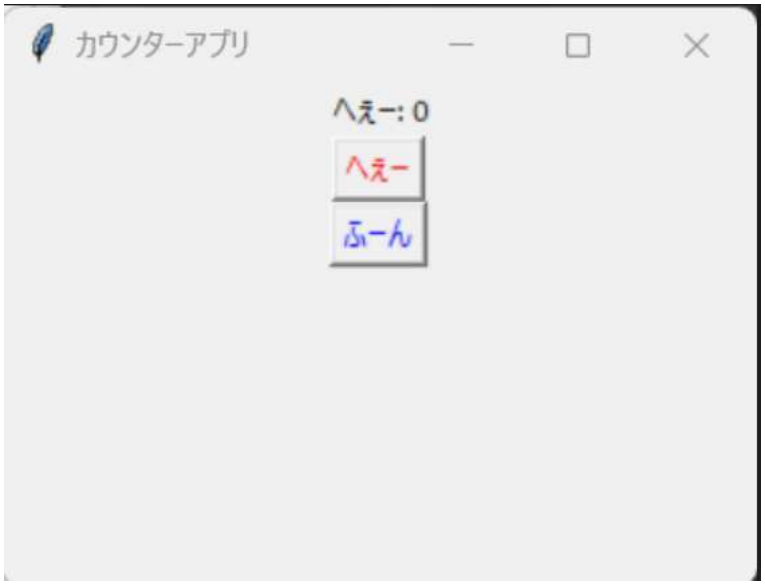
ウィジェットの生成，配置，動作の設定

クラスの定義を見ていきましょう．

- 初期化する__init__メソッドでその大部分が構成されています．引数には，おきまりのselfに加えて，tkのメインウィンドウのオブジェクト（root）を受け取るmasterが定義され，最初の行で早速self.master=masterとして，自分自身のインスタンス変数としてのmasterに割り当てています．ちょっとややこしいかもしれませんが，式の左のself.masterはインスタンス変数，右のmasterは関数定義の際の引数としての変数です．
- 続けて，self.masterのメソッドとして定義されている，タイトルを設定するtitle，ウィンドウの大きさを設定するgeometryメソッドを実行しています．
- 次に，カウンタアプリとしての状態であるカウント数を保持するインスタンス変数としてself.countを定義しています．
- その後，いよいよ部品を生成しています．tk.Label(self.master, text="へぇー: 0")は，tkに定義されているラベルを表すクラスであるLabelのオブジェクトをいくつかの引数を指定して生成し，self.label，つまり，このクラスのインスタンス変数としてのlabelに割り当てています．最初の引数では，この部品の親のウィジェットとしてウィンドウオブジェクトであるself.masterを指定しています．次の引数textはラベルとして表示される文字列を指定します．ウェジェットの生成時にはたくさんの引数が定義されているので，このように設定に必要な「text="へぇー: 0"」などとして，引数名と値のペアのみを列挙します．
- self.label.pack()は，作成したラベルの具体的な配置方法を指定しつつ親ウィジェットに配置するメソッドです．pack()は，このようにとくに指定がない場合は，実行されるごとに上から順番に部品を配置します．
- 同様にして，tk.Button(...)でボタンを表すオブジェクトを生成し，self.buttonに割り当てます．引数fgはフォアグラウンドの略で，文字列でボタンの文字色を指定します．ボタンは押されたときの動作を指定する必要があるため，command引数で，押されたときに実行される関数を指定します．このようなあるイベントが生じたときに呼び出される関数を「コールバック関数」と呼ぶことがあります．
- 今回はその関数として下のself.incrementメソッドが指定されています．まずカウンタの値を1つ増やし，次に，self.labelに定義されている状態を更新するメソッドであるconfigを利用して，引数としてtextに更新後の値を与えて実行しています．

練習 1

このプログラムに，押すとカウンタが1減る「ふーん」ボタンを文字を青色にしてつけてみましょう．



さまざまなウィジェットとレイアウト

これは最小限の例でしたが，皆さんが見知ったウィンドウの部品がそれぞれクラスとして定義されていて，生成（設定）・配置を行うことで利用することができます．

例えば，以下は主要なウィジェットと生成時の重要な引数，それらが持つ重要なメソッドの利用例等をまとめたものです． 見ての通り非常にたくさんの種類や引数，メソッドなどが定義されています．これらはぜんぶ覚えるのではなくて，適宜必要に応じてどんなものが使えるか，自分で調べたり，VSCodeの補完機能を活用したりして使いこなせるようになることが重要です．

ウィジェット	説明	主要メソッドと引数	使用例
Label	テキストや画像を表示	`Label(master, text="", image=None, font=(), fg="black", bg="white")`: ラベルを生成 `config(text="新しいテキスト")`: テキストを変更 `config(image=img)`: 画像を設定	`label = tk.Label(root, text="Hello", font=("Arial", 12))` `label.config(text="新しいテキスト")`
Button	クリック可能なボタン	`Button(master, text="", command=None, state="normal")`: ボタンを生成 `config(state="disabled")`: ボタンを無効化 `config(command=new_func)`: コマンドを変更	`button = tk.Button(root, text="Click", command=func)` `button.config(state="disabled")`

Entry	一行のテキスト入力フィールド	`Entry(master, width=20, show="", textvariable=None)`: 入力フィールドを生成
		`get()`: 入力内容を取得
		`insert(index, string)`: テキストを挿入
		`delete(first, last)`: テキストを削除
		`entry.bind("イベント名", command=実行関数)`: この入力フィールドに関するイベントと関数をバインド
			`entry = tk.Entry(root, width=30)`
			`text = entry.get()`
			`entry.insert(0, "Hello")`
			`entry.delete(0, tk.END)`
			`entry.bind('<Return>', self.update_count)`

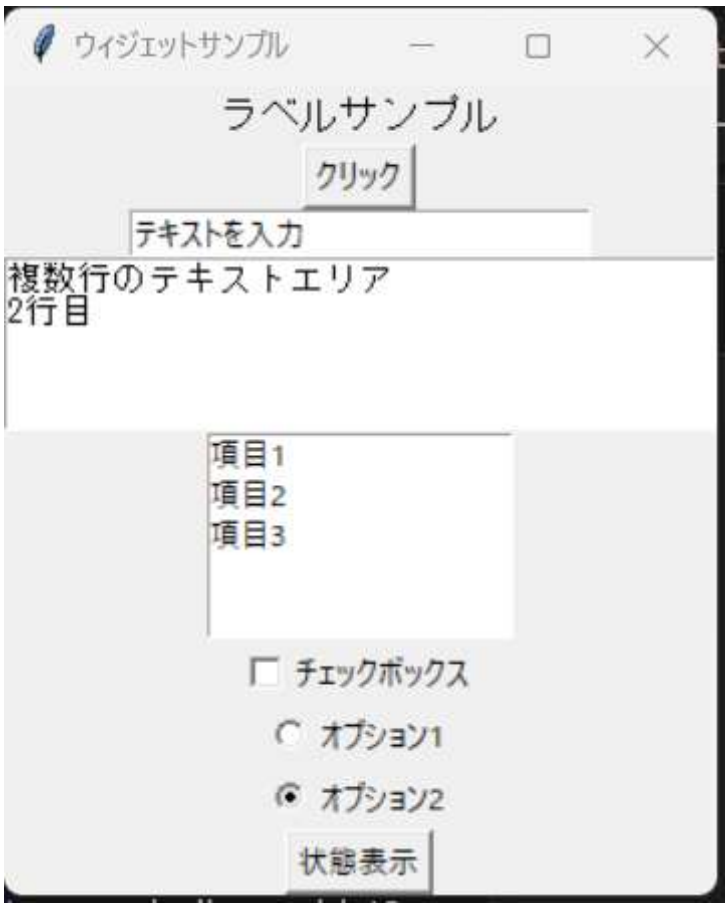
ウィジェット	説明	主要メソッドと引数	使用例
Text	複数行のテキスト入力領域	Text(master, width=40, height=10, wrap="word"): テキストエリアを生成 get(start, end): テキストを取得 insert(index, string): テキストを挿入 delete(start, end): テキストを削除	text = tk.Text(root, width=40, height=10) content = text.get("1.0", tk.END) text.insert(tk.END, "新しい行\n") text.delete("1.0", "2.0")
Listbox	選択可能な項目のリスト	Listbox(master, height=10, selectmode="single"): リストボックスを生成 insert(index, *elements): 項目を追加 delete(first, last): 項目を削除 get(first, last): 項目を取得 curselection(): 選択された項目のインデックスを取得	listbox = tk.Listbox(root, height=5) listbox.insert(tk.END, "項目1", "項目2") selected = listbox.curselection()
Checkbutton	オン/オフの状態を持つボタン	Checkbutton(master, text="", variable=None, command=None): チェックボタンを生成 select(): チェックを入れる deselect(): チェックを外す toggle(): 状態を切り替える	var = tk.BooleanVar() cb = tk.Checkbutton(root, text="オプション", variable=var) cb.select()
Radiobutton	複数の選択肢から一つを選ぶボタン	Radiobutton(master, text="", variable=None, value=None, command=None): ラジオボタンを生成 select(): このボタンを選択状態にする deselect(): 選択を解除する	var = tk.IntVar() rb1 = tk.Radiobutton(root, text="選択1", variable=var, value=1) rb2 = tk.Radiobutton(root, text="選択2", variable=var, value=2) rb1.select()

以下は，ウィジェットを使ったサンプルです．

```
import tkinter as tk

class SampleApp:
    def __init__(self, master):
        self.master = master
        master.title("ウィジェットサンプル")

        # Label
        self.label = tk.Label(master, text="ラベルサンプル", font=("Arial", 12))
        self.label.pack()

        # Button
        self.button = tk.Button(master, text="クリック", command=self.button_click)
```

```
        self.button.pack()

        # Entry
        self.entry = tk.Entry(master, width=30)
        self.entry.pack()
        self.entry.insert(0, "テキストを入力")
        self.entry.bind('<Return>', self.update_label)

        # Text
        self.text = tk.Text(master, width=40, height=5)
        self.text.pack()
        self.text.insert(tk.END, "複数行のテキストエリア\n2行目")

        # Listbox
        self.listbox = tk.Listbox(master, height=5)
        self.listbox.pack()
        for item in ["項目1", "項目2", "項目3"]:
            self.listbox.insert(tk.END, item)

        # Checkbutton (IntVar/BooleanVar不使用)
        self.check_state = False
        self.checkbutton = tk.Checkbutton(master, text="チェックボックス", command=self.toggle
        self.checkbutton.pack()

        # Radiobutton (IntVar/BooleanVar不使用)
        self.radio_state = 0
        self.radio1 = tk.Radiobutton(master, text="オプション1", value=1, command=lambda: sel
        self.radio2 = tk.Radiobutton(master, text="オプション2", value=2, command=lambda: sel
        self.radio1.pack()
        self.radio2.pack()

        # 状態表示ボタン
        self.state_button = tk.Button(master, text="状態表示", command=self.show_state)
        self.state_button.pack()

    def button_click(self):
        tk.messagebox.showinfo("ボタン", "ボタンがクリックされました")

    def update_label(self, event):
        new_text = self.entry.get()
        self.label.config(text=new_text)

    def toggle_check(self):
        self.check_state = not self.check_state
        print(f"チェックボックス: {'オン' if self.check_state else 'オフ'}")
```

```
    def set_radio(self, value):
        self.radio_state = value
        print(f"ラジオボタン: オプション{value}")

    def show_state(self):
        message = f"チェックボックス: {'オン' if self.check_state else 'オフ'}\n"
        message += f"ラジオボタン: オプション{self.radio_state}\n"
        message += f"リストボックス選択: {self.listbox.curselection()}\n"
        message += f"テキストエリア内容: {self.text.get('1.0', tk.END).strip()}"
        tk.messagebox.showinfo("現在の状態", message)

root = tk.Tk()
app = SampleApp(root)
root.mainloop()
```



また， 部品の配置決めるレイアウトについても， pack以外にも様々な方法があります．

メソッド	説明	主要パラメータ
pack()	ウィジェットを親ウィジェット内に順次配置	side: 配置する側（top, bottom, left, right） fill: ウィジェットの拡大方向（x, y, both, none） expand: 親の余白スペースを埋めるかどうか（True/False）
grid()	ウィジェットを行と列の格子状に配置	row: 配置する行番号 column: 配置する列番号 padx: 水平方向のパディング pady: 垂直方向のパディング sticky: セル内でのウィジェットの位置（n, s, e, w, ne, nw, se,

メソッド	説明	主要パラメータ
		sw）
place()	ウィジェットを絶対位置または相対位置に配置	x: 絶対X座標 y: 絶対Y座標 relx: 相対X座標（0.0 ～ 1.0） rely: 相対Y座標（0.0 ～ 1.0） anchor: ウィジェットの基準点（n, s, e, w, center, ne, nw, se, sw）

（最初の最小限のウィンドウアプリのコードにいれてみてください．）

[pack_example]:

```
label = tk.Label(root, text="上部")
label.pack(side="top", fill="x")
button = tk.Button(root, text="下部")
button.pack(side="bottom", fill="x")
text = tk.Text(root)
text.pack(fill="both", expand=True)
```

[grid_example]:

```
label = tk.Label(root, text="名前:")
label.grid(row=0, column=0, padx=5, pady=5)
entry = tk.Entry(root)
entry.grid(row=0, column=1, padx=5, pady=5)
button = tk.Button(root, text="送信")
button.grid(row=1, column=0, columnspan=2)
```

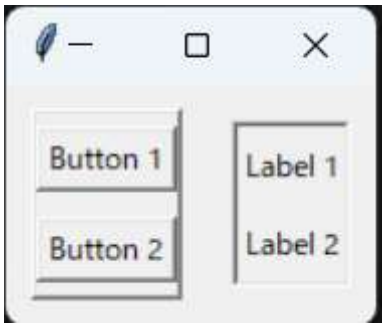
[place_example]:

```
label = tk.Label(root, text="絶対位置")
label.place(x=10, y=10)
button = tk.Button(root, text="相対位置")
button.place(relx=0.5, rely=0.5, anchor="center")
text = tk.Text(root, width=20, height=3)
text.place(relx=1.0, rely=1.0, anchor="se", x=-10, y=-10)
```

この後も時々出てきますが，パラメータに与えられる値のいくつかは，“tk.LEFT”なら“左に配置”等のように定数が定義されています．これらも適宜調べて活用できます．

また，Frameクラスは部品をまとめるレイアウト枠を表します．Frameクラスのオブジェクトを配置し，それを親としてその中にウィジェットを配置できます．以下は簡単な例です．

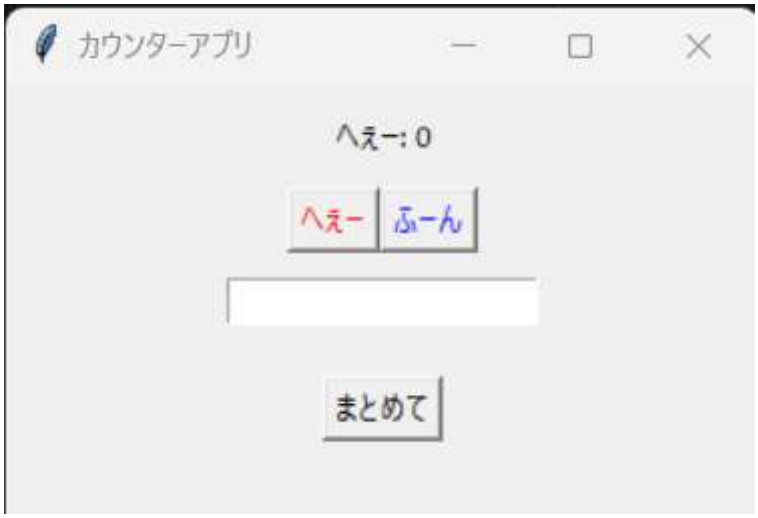
```python
import tkinter as tk

root = tk.Tk()
root.title("Frame Example")

# 左フレームの作成と配置
left_frame = tk.Frame(root, bd=2, relief=tk.RAISED)
left_frame.pack(side=tk.LEFT, padx=10, pady=10)

# 右フレームの作成と配置
right_frame = tk.Frame(root, bd=2, relief=tk.SUNKEN)
right_frame.pack(side=tk.RIGHT, padx=10, pady=10)

# 左フレームにボタンを追加
button1 = tk.Button(left_frame, text="Button 1")
button1.pack(pady=5)
button2 = tk.Button(left_frame, text="Button 2")
button2.pack(pady=5)

# 右フレームにラベルを追加
label1 = tk.Label(right_frame, text="Label 1")
label1.pack(pady=5)
label2 = tk.Label(right_frame, text="Label 2")
label2.pack(pady=5)

root.mainloop()
```



練習 2

上の情報を参考にして，また，必要であれば適宜検索などで調べて，カウンタアプリを以下のように改変しましょう．

- へぇーボタンとふーんボタンを横に並べる
- 一行入力のテキストフィールドであるEntryオブジェクトと「まとめて」ボタンを追加する．フィールドに整数値が入力され，「まとめて」ボタンが押されると整数値の分だけカウンタが増減する．



キャンバスとイベント処理

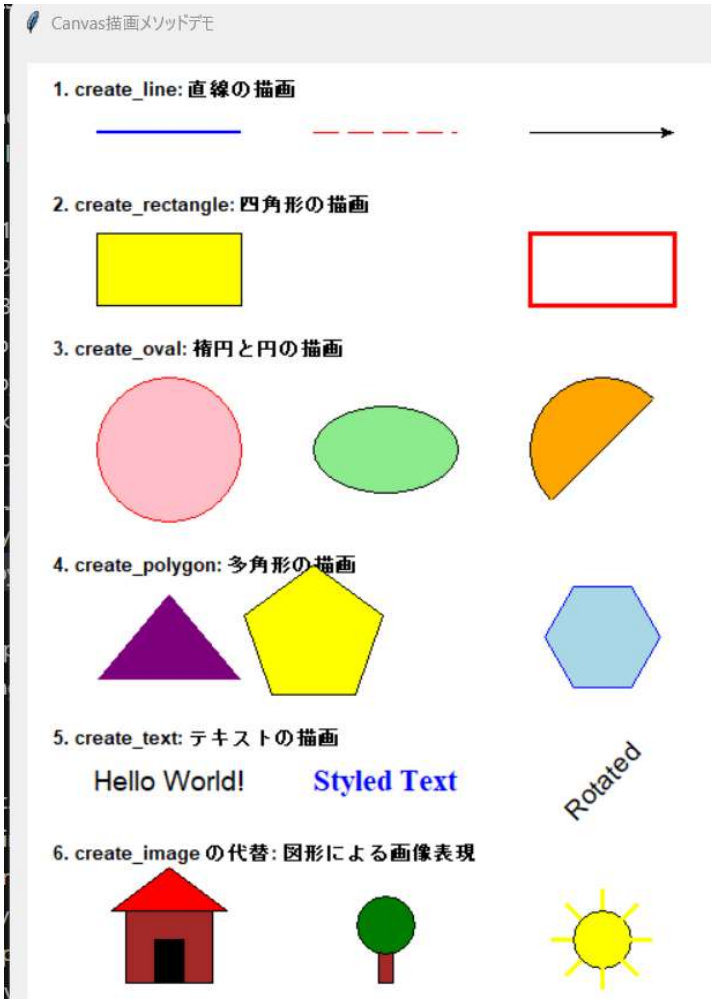
最後に，ウィンドウに自由に描画するキャンバスクラスと，ユーザからの操作の情報であるイベントを受け取るための処理を紹介します．
これまでのウィジェットと同様にCanvasという文字通りお絵描きするためのキャンバスを表すウィジェットがあります．プログラミングの授業で簡単なゲームをつくる際によく利用されるので，ここでもその基本を取り上げます．
ちょっと長いですが以下は，大きなキャンバスオブジェクトを一つ作成し，その上にいろいろな図形を描画するサンプルです．

```
import tkinter as tk
import math
class CanvasMethodsDemo:
    def __init__(self):
        # メインウィンドウの作成
        self.root = tk.Tk()
        self.root.title("Canvas描画メソッドデモ")
        # キャンバスの作成
        self.canvas = tk.Canvas(self.root, width=600, height=800, bg="white")
        self.canvas.pack(padx=10, pady=10)
        # 各描画メソッドのデモを実行
        self.demonstrate_methods()
        # メインループ開始
        self.root.mainloop()
    def demonstrate_methods(self):
        y_offset = 20  # Y座標のオフセット
        # 1. create_line: 直線を描画
        self.add_title("1. create_line: 直線の描画", y_offset)
        # 通常の直線
        self.canvas.create_line(50, y_offset+30, 150, y_offset+30, width=2, fill="blue")
        # 破線
        self.canvas.create_line(200, y_offset+30, 300, y_offset+30, dash=(5,2), fill="red")
```

```
        # 矢印付き直線
        self.canvas.create_line(350, y_offset+30, 450, y_offset+30, arrow=tk.LAST)
        # 2. create_rectangle: 四角形を描画
        y_offset += 80
        self.add_title("2. create_rectangle: 四角形の描画", y_offset)
        # 塗りつぶし四角形
        self.canvas.create_rectangle(50, y_offset+20, 150, y_offset+70,
                                     fill="yellow", outline="black")
        # 枠のみの四角形
        self.canvas.create_rectangle(350, y_offset+20, 450, y_offset+70,
                                     outline="red", width=3)
        # 3. create_oval: 楕円を描画
        y_offset += 100
        self.add_title("3. create_oval: 楕円と円の描画", y_offset)
        # 円
        self.canvas.create_oval(50, y_offset+20, 150, y_offset+120,
                                fill="pink", outline="red")
        # 楕円
        self.canvas.create_oval(200, y_offset+40, 300, y_offset+100,
                                fill="light green")
        # 弧
        self.canvas.create_arc(350, y_offset+20, 450, y_offset+120,
                               start=45, extent=180, fill="orange")
        # 4. create_polygon: 多角形を描画
        y_offset += 150
        self.add_title("4. create_polygon: 多角形の描画", y_offset)
        # 三角形
        self.canvas.create_polygon(50, y_offset+80, 100, y_offset+20,
                                   150, y_offset+80, fill="purple")
        # 星型
        points = []
        for i in range(5):
            points.extend([
                200 + 50*math.cos(2*math.pi*i/5 - math.pi/2),
                y_offset+50 + 50*math.sin(2*math.pi*i/5 - math.pi/2)
            ])
        self.canvas.create_polygon(points, fill="yellow", outline="black")
        # 六角形
        hexagon_points = []
        for i in range(6):
            hexagon_points.extend([
                400 + 40*math.cos(2*math.pi*i/6),
                y_offset+50 + 40*math.sin(2*math.pi*i/6)
            ])
        self.canvas.create_polygon(hexagon_points, fill="light blue", outline="blue")
        # 5. create_text: テキストを描画
```

```
        y_offset += 120
        self.add_title("5. create_text: テキストの描画", y_offset)
        # 通常のテキスト
        self.canvas.create_text(100, y_offset+30, text="Hello World!",
                                font=("Arial", 14))
        # スタイル付きテキスト
        self.canvas.create_text(250, y_offset+30, text="Styled Text",
                                font=("Times", 16, "bold"), fill="blue")
        # 回転テキスト
        self.canvas.create_text(400, y_offset+30, text="Rotated",
                                font=("Arial", 14), angle=45)
        # 6. create_image: 画像の代わりに図形で表現
        y_offset += 80
        self.add_title("6. create_image の代替: 図形による画像表現", y_offset)
        # 簡単な家の絵
        self.draw_house(100, y_offset+60)
        # 簡単な木の絵
        self.draw_tree(250, y_offset+60)
        # 簡単な太陽の絵
        self.draw_sun(400, y_offset+60)
    def add_title(self, text, y_offset):
        """セクションタイトルを追加"""
        self.canvas.create_text(20, y_offset, text=text,
                                anchor=tk.W, font=("Arial", 10, "bold"))
    def draw_house(self, x, y):
        """家の絵を描画"""
        # 家の本体
        self.canvas.create_rectangle(x-30, y-20, x+30, y+30, fill="brown")
        # 屋根
        self.canvas.create_polygon(x-40, y-20, x+40, y-20, x, y-50,
                                   fill="red", outline="black")
        # ドア
        self.canvas.create_rectangle(x-10, y, x+10, y+30, fill="black")
    def draw_tree(self, x, y):
        """木の絵を描画"""
        # 幹
        self.canvas.create_rectangle(x-5, y, x+5, y+30, fill="brown")
        # 葉
        self.canvas.create_oval(x-20, y-30, x+20, y+10, fill="green")
    def draw_sun(self, x, y):
        """太陽の絵を描画"""
        # 本体
        self.canvas.create_oval(x-20, y-20, x+20, y+20, fill="yellow")
        # 光線
        for i in range(8):
            angle = i * math.pi/4
```

```
            self.canvas.create_line(
                x + 20*math.cos(angle), y + 20*math.sin(angle),
                x + 35*math.cos(angle), y + 35*math.sin(angle),
                fill="yellow", width=3
            )
# デモの実行
if __name__ == "__main__":
    CanvasMethodsDemo()
```



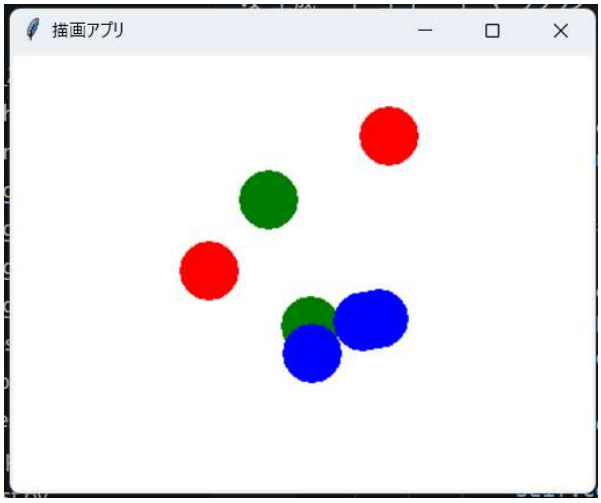
このコードもアプリとしてクラスを定義する形式になっていてやや難しそうにも見えますが，よく見ると単純です．お決まりの__init__メソッドで，self.canvas = tk.Canvas(self.root, width=600, height=800, bg="white")を実行して，キャンバスクラスのオブジェクトをself.canvasに割り当てています．

その後，別に定義したdemonstrate_methodsメソッドを実行してその中でいろいろ描画しています．いろいろすぎるかもしれませんが，基本は「self.canvas.(図形に対応する作成メソッド)」の形式です．各メソッド使用方法は，検索してもよいですし，VSCodeではメソッドにマウスポインタをあてると引数などの説明がポップアップしますので，試してみてください．

次に、テキスト入力フィールドと描画機能を組み合わせて，キー入力とマウス操作に関するイベント処理を利用した簡単な例を見てみましょう．この例では，マウスでクリックした場所に円を描画

します．円の色をキーボードのr, g, bで変更できます．

```
import tkinter as tk

class DrawingApp:
    def __init__(self, master):
        self.master = master
        self.master.title("描画アプリ")

        self.canvas = tk.Canvas(self.master, width=400, height=300, bg="white")
        self.canvas.pack()

        self.canvas.bind("<Button-1>", self.draw_circle)
        self.master.bind("<KeyPress>", self.change_color)

        self.color = "red"

    def draw_circle(self, event):
        x, y = event.x, event.y
        r = 20
        self.canvas.create_oval(x-r, y-r, x+r, y+r, fill=self.color, outline="")

    def change_color(self, event):
        if event.char == "r":
            self.color = "red"
        elif event.char == "g":
            self.color = "green"
        elif event.char == "b":
            self.color = "blue"

root = tk.Tk()
app = DrawingApp(root)
root.mainloop()
```



重要な点は，

self.canvas.bind("<Button-1>", self.draw_circle)

の部分です．bindメソッドはそのオブジェクトに関して起きるイベントと，それに対応して実行されるコールバック関数を関連付けます．"<Button-1>”はマウスの左ボタンが押されたというイベントを示します．これが押されたら，ボタンクラスの場合と同様に，self.draw_circleが実行されることを示しています．

```
def draw_circle(self, event):
    x, y = event.x, event.y
    r = 20
    self.canvas.create_oval(x-r, y-r, x+r, y+r, fill=self.color, outline="")
```

このように，引数にeventを指定することで，関数内で発生したイベントに関する情報を得ることができます．この場合，event.x，event.yでマウスがクリックされたx, y位置（ただしウィンドウ領域の左上が(0, 0))が得られるので，これを利用してクリックした場所に円を作成しています．

self.master.bind("<KeyPress>", self.change_color)

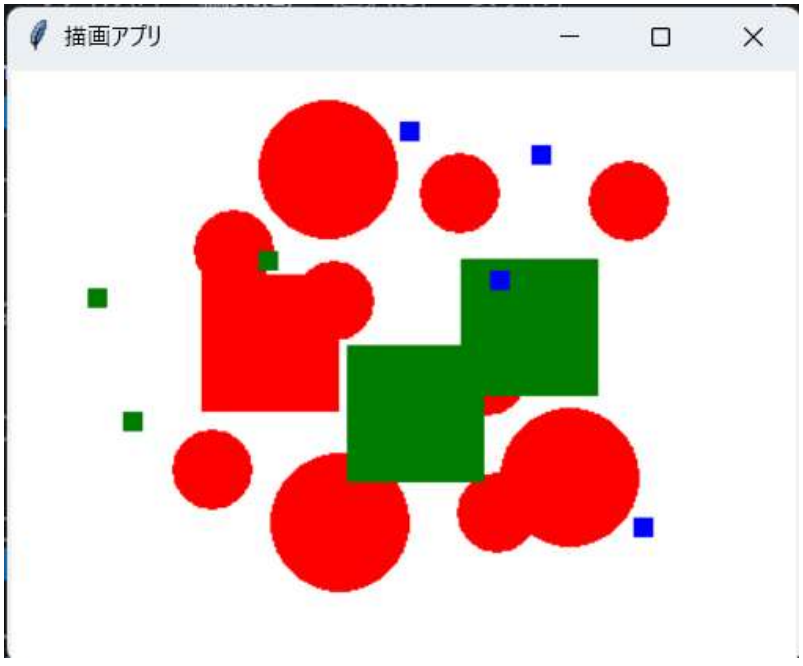
も同様ですが，event.charで押されたキーの文字を参照しています．

"<Button-1>”や"<KeyPress>"には様々なものと利用法があります．調べてみましょう．

練習 3

上のアプリのコードに追記して，次の機能を加えてみましょう．

- 右クリックでクリックした場所に四角を描く
- キーボードの「u」または「d」を押すと描画する円のサイズが増減する



（GUIの利活用に関する課題）