# プログラミングにおける生成 AI の活用方法

## 1. はじめに

ChatGPT などの生成 AI は、プログラムを作成することが可能である。実際に動かすことで正しいかどうかの判断ができるので、文章作成などの事実確認が必要な分野と異なり、活用がしやすい。そのため、プログラミングにおいて生成 AI を活用することは有効だと感じ、今回の研究に至った。

## 2. 方法

今回の実験では、Microsoft Copilot を使用した。
以下、特に記載がなければ「生成 AI」は Microsoft Copilot を指す。

Python で、一つのファイルで構成される Todo リストを作成。以下の 2 つを比較する。

・自力で作成したもの。参考として、総制作時間は 6 時間。
・自作プログラム全体を生成 AI に読み込ませ、100 文字以内で説明させた文章をプロンプトとして作成。
**「以下のようなプログラムを Python で作成してください。このコードは、SQLite データベースを使用して ToDo リストを管理するための Tkinter ベースの GUI アプリケーションです。タスクの追加、完了、過去のタスクの表示ができます。」**
このプロンプトを、生成 AI に入力して生成されたもの。

## 3. 結果

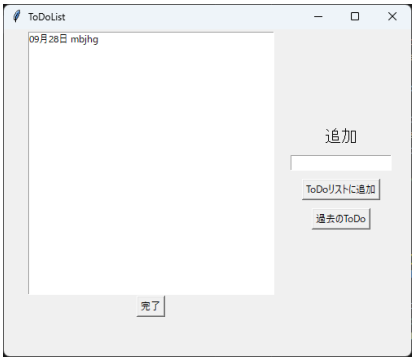


図 1

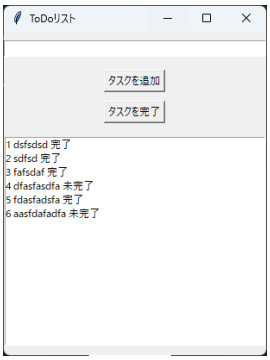


図 2

図 1 が自作したプログラム、図 2 が生成されたプログラムである。

プログラム内部の SQL 文は、自動的に生成されている。そのため、テーブルの形式なども自動的に作成されている。プロンプトで具体的な指定を行わず、「SQLite データベースを使用して」とだけ書いたのでその影響と見られる。（図 3）
両方ともにバグやエラーはない。しかし、ChatGPT で生成されたプログラムには、自作版にはない try-except の例外処理が含まれており、よりエラーが起きにくくなっている。
画面部品のレイアウトに関する処理が一切ない。そのため、画面部品は上から順に並べられているだけである。おそらく、プロンプトにそのような記述が一切ない影響だと思われる。

```python
# データベースのセットアップ
conn = sqlite3.connect('todo2.db')
c = conn.cursor()
c.execute('''CREATE TABLE IF NOT EXISTS tasks
             (id INTEGER PRIMARY KEY, task TEXT, status TEXT)''')
conn.commit()
```

図 3 生成されたプログラムでは、初回起動時にテーブルが生成されるようになっている。
（注:'todo2.db'は後から指定したもの。生成時は'todo.db'となっていたが、自作プログラムとの区別のため変更）

自作版においてすでに終了したタスクは、「過去の Todo」というボタン（図 1 参照）を押して別画面で出るようにしているが、生成版では残ってしまう。実際に図 2 には、「完了」というラベルの付いたタスクが残っている。プロンプト内の「過去のタスクの表示」という記述から、このような仕様にしたと考えられる。

```python
# 追加処理
def addList(event):
    task = add.get()

    dt_now = datetime.datetime.now()
    date = dt_now.strftime('%m月%d日')

    if ( task == '' ):
        return None

    sql = 'insert into todo (date, content, flag) values ("'+ date +'","'+ task +'", 0)'

    c.execute(sql)
    conn.commit()

    sql = 'select id, date, content, flag from todo where content = "'+ task +'" '
    allist = c.execute(sql)

    for row in allist:
        if ( row != '' ):
            tmp = f'{row[2]} 登録日時:{row[1]}'
            ListBox.insert(END, tmp)
            d[tmp] = row[2]

    # ListBox.insert(END, task)

    add.delete(0, END)
```

2 つのプログラムは仕様が異なる部分が多いので、処理が似ている「タスクの追加」部分の比較。
右図（図 4）は自作したプログラム、下図（図 5）は生成されたプログラムである。

```python
# タスクを追加する関数
def add_task():
    task = task_entry.get()
    if task:
        c.execute('INSERT INTO tasks (task, status) VALUES (?, ?)', (task, '未完了'))
        conn.commit()
        task_entry.delete(0, tk.END)
        display_tasks()
    else:
        messagebox.showwarning("警告", "タスクを入力してください")
```

## まとめ

・自作したプログラムとの比較は、作成者の能力などの面もあり単純に比較することは難しい。しかし処理の一つ一つを見ると、AI で生成したもののほうが優れている部分は存在した。
・SQLite に関する AI が作成した処理を見るに、プログラム上の構造に指定が必要な場合、明記したほうがいい。
・個人的には、AI は書かれたことしか実装しないという印象を持った。逆に言えば、詳細な設計を事前にまとめた上で指示を出せば、一回で完成すると考えられる。
・全体像を指示するのは難しい場面も多い。また生成したプログラムには間違いが含まれている可能性もあるので、一度に大きなプログラムを作成するとエラーの発見も難しい。そのため、部分的にプログラムを作成させるといった補助的な使い方には向いていると考えられる。
・今回は一つのファイルで完結するプログラムだが、他のより大規模なプロジェクトだと複数のディレクトリ・ファイルを扱う。その全体像を把握しながら質問に答えたり、プログラムを生成したりする AI があれば、開発を効率的に進められるのではと感じた。

**SQLite**:データベース管理システム。他のものと違い、単一のファイルのみを使用する軽量な仕様。
**Tkinter**:Python で GUI アプリを制作する際に使用するライブラリ。