

プロジェクト演習 テーマD

第9回

担当：CS学部 講師 伏見卓恭
連絡先：fushimity@edu.teu.ac.jp

授業の流れ

- 第 8回:実験環境の構築 / Python, Gitの復習 / CUIゲームの開発
- 第 9回:Tkinterの基礎
- 第10回:TkinterによるGUIゲーム開発
- 第11回:PyGameの基礎
- 第12回:PyGameによるゲーム開発
- 第13回:ゲーム開発演習
- 第14回:成果発表

本日のお品書き

1. 前回の振り返り
2. 実験環境の構築 (Tkinter)
3. Tkinterの基礎
4. Tkinterを使ってGUIアプリの開発

実験環境の構築

mainブランチになっていることを確認する

ProjExD2022/ex02/フォルダを作成する

tkinter (Tkインタフェース)

- tkinterとは, PythonでGUIを組むことのできるツールキット

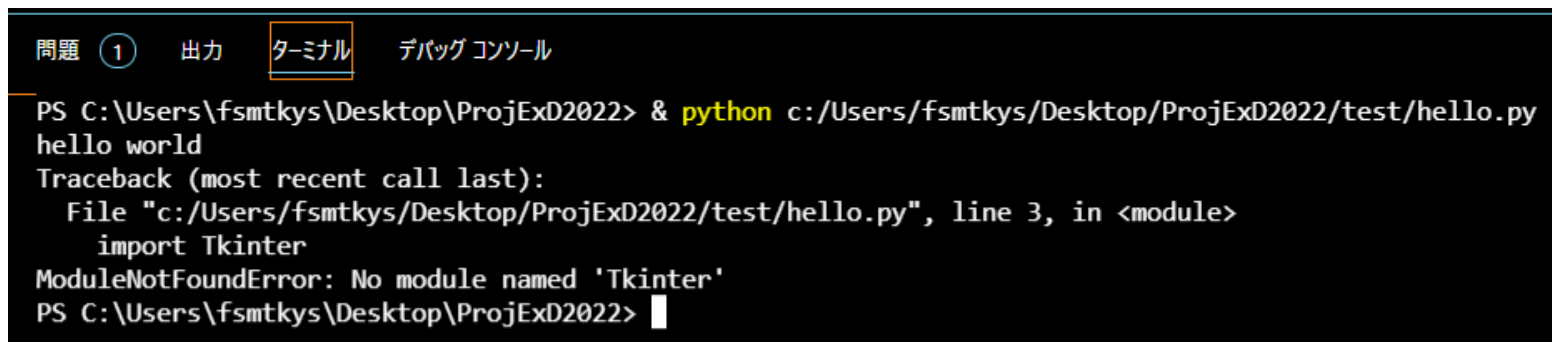


```
1 print("hello world")
2
3 import tkinter as tk
4
5 root = tk.Tk()
6 root.mainloop()
```

Q:何をしている?

Q:何をしている?

- ModuleNotFoundErrorが出る場合は,
tkinterモジュールをインストールする



```
PS C:\Users\fsmtkys\Desktop\ProjExD2022> & python c:/Users/fsmtkys/Desktop/ProjExD2022/test/hello.py
hello world
Traceback (most recent call last):
  File "c:/Users/fsmtkys/Desktop/ProjExD2022/test/hello.py", line 3, in <module>
    import Tkinter
ModuleNotFoundError: No module named 'Tkinter'
PS C:\Users\fsmtkys\Desktop\ProjExD2022>
```

tkinterモジュールのインストール

- Anaconda prompt, または, コマンドプロンプトで, 以下を入力する

```
(ProjExD) C:¥Users¥fsmtkys>conda install -c anaconda tk
```

- インストールされたものを確認する

```
(ProjExD) C:¥Users¥fsmtkys>conda list
# packages in environment at C:¥Users¥fsmtkys¥anaconda3¥envs¥ProjExD:
#
# Name                                Version                                Build    Channel
:
tk                                    8.6.11                                h2bbff1b_0  Anaconda
:
```

参考URL :

<https://java-beginner.com/python-tkinter-hello-world/>

Tkインタフェースの基礎

参考URL : <https://docs.python.org/ja/3.7/library/tkinter.html>

tkinterで作れるウィジェット

- Tk() : ウィンドウ
- Label() : 文字列ラベル
- Button() : ボタン
- Canvas() : キャンバス
- Entry() : テキスト入力欄
- Text() : 複数行のテキスト入力欄
- Checkbutton() : チェックボタン (複数選択可)
- Radiobutton() : ラジオボタン (単一選択のみ)

Tkクラス

- インスタンスメソッド
 - title(タイトル) : ウィンドウのタイトルを設定する
 - geometry(サイズ) : ウィンドウのサイズ"幅x高さ"を設定する
 - resizable(横方向, 縦方向) : サイズ変更可能True or 不可能False
 - mainloop() : ウィンドウを表示する

```
3 import tkinter as tk
4
5 root = tk.Tk()
6 root.title("おためしか")
7 root.geometry("500x200")
8 root.mainloop()
```

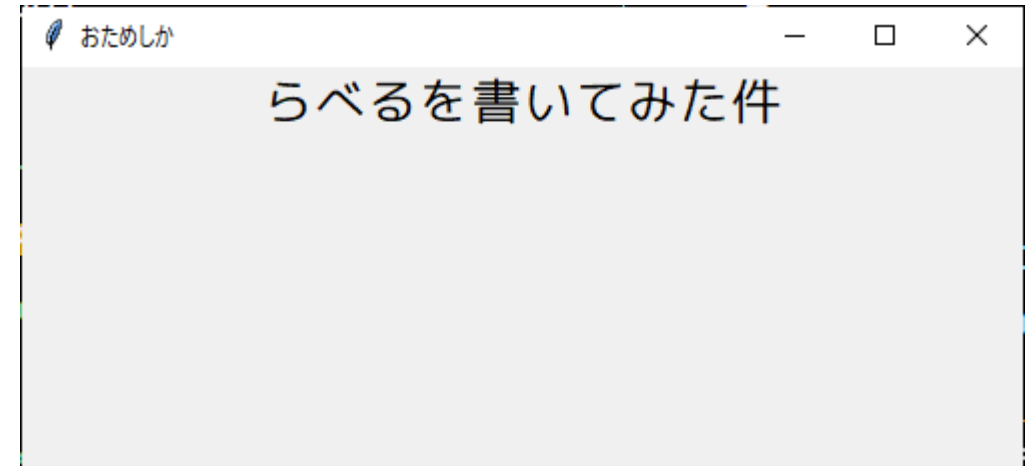


※プログラム実行により開いたウィンドウは、その都度閉じるようにしよう ↑ ↑ ↑

Labelクラス

- コンストラクタに指定する引数
 - 親ウィンドウ
 - text：ラベルの文字列
 - font：(フォントタイプ, フォントサイズ)

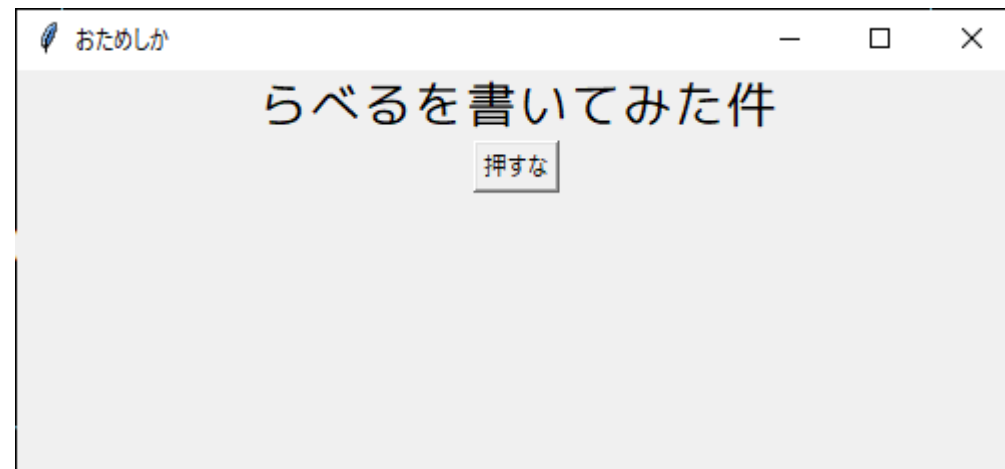
```
5 root = tk.Tk()
6 root.title("おためしか")
7 root.geometry("500x200")
8
9 label = tk.Label(root,
10 |         |         |         | text="らべるを書いてみた件",
11 |         |         |         | font=("Ricty Diminished", 20)
12 |         |         |         | )
13 label.pack()
14
15 root.mainloop()
```



Buttonクラス

- コンストラクタに指定する引数
 - 親ウィンドウ
 - text : ボタンの文字列
 - font : (フォントタイプ, フォントサイズ)
 - bg : 背景色
 - fg : 文字の色
 - command : クリックした際の反応

```
15 button = tk.Button(root, text="押すな")
16 button.pack()
17
18 root.mainloop()
```



Canvasクラス

- コンストラクタに指定する引数
 - 親ウィンドウ
 - width：キャンバスの幅
 - height：キャンバスの高さ
 - bg：背景色
- インスタンスメソッド
 - create_image(x座標, y座標, image=PhotoImageクラスのインスタンス)
 - create_line(座標列, fill=色, width=線の太さ)
 - create_rectangle(座標列, fill=塗り色, outline=枠線の色, width=枠線の太さ)
 - create_oval(rectangleと同様)
 - create_polygon(rectangleと同様)

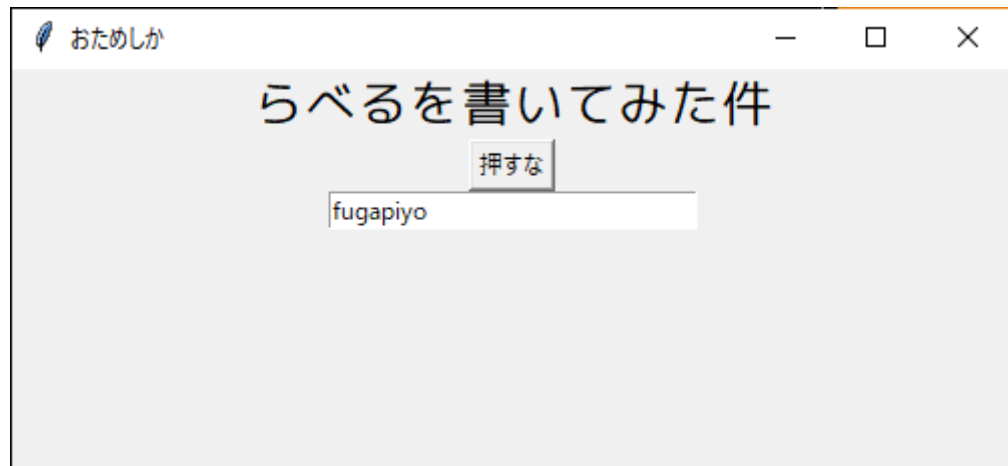
PhotoImageクラス

- コンストラクタに指定する引数
 - `file` : 画像ファイルのパス

Entryクラス

- コンストラクタに指定する引数
 - width : 入力文字数（半角）で入力欄の幅を指定
- インスタンスメソッド
 - get(開始位置, 終了位置) : 入力欄の文字列を取得
 - insert(位置, 文字列) : 位置に文字列を挿入
 - 位置を入力欄の末尾にしたい場合 : tkinter.END
 - delete(開始位置, 終了位置) : 入力欄の文字列を削除

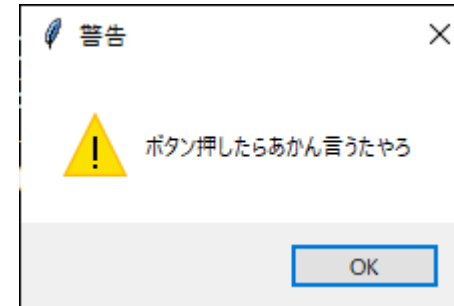
```
22 entry = tk.Entry(width=30)
23 entry.insert(tk.END, "fugapiyo")
24 entry.pack()
25
26 root.mainloop()
```



メッセージボックス

- tkinter.messageboxモジュールをimportする
- messageboxモジュールの関数
 - showinfo(タイトル, メッセージ)
 - showwarning(タイトル, メッセージ) →
 - showerror(タイトル, メッセージ)
 - askyesno(タイトル, メッセージ)
 - askokcancel(タイトル, メッセージ)

```
3 import tkinter as tk
4 import tkinter.messagebox as tkm
```



```
6 ✓ def button_click():
7     |     tkm.showwarning("警告", "ボタン押したらあかん言うたやろ")
```

← クリック時に発動する関数を定義

```
19 button = tk.Button(root, text="押すな", command=button_click)
20 button.pack()
```

← 関数をボタンに設定

ウィジェットの配置

- 各ウィジェットを表すインスタンス変数のメソッドを利用する
 - `place(座標)` : 親ウィンドウ上の配置座標を設定する
 - `pack(方向)` : 指定した方向 (デフォルト : 上) から順に配置する
 - `grid(行数, 列数)` : エクセルのような仮想的な表において行数, 列数を指定して配置する
- 数学の直交座標系と異なり, 縦方向は下に行くほど値が大きい
→ 左上が $(0,0)$

※gridのわかりやすいサイト :

https://imaging-solution.net/program/python/tkinter/widget_layout_grid/

イベントの紐づけ (bind)

- マウスの { 左, 右 } クリック, キーの押下などのユーザによるイベントが発生した時に実行される関数を, イベントと紐づける

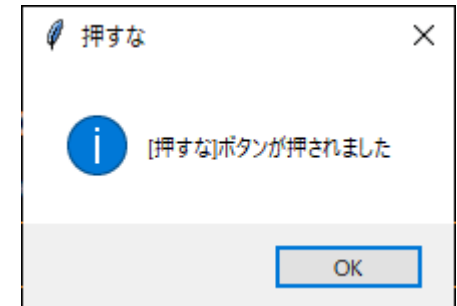
`widget.bind("<イベント>", イベント発生時に呼び出される関数名)`

```
19 button = tk.Button(root, text="押すな", command=button_click)
20 button.bind("<1>", button_click)
21 button.pack()
```

← "<1>"=左クリック が起きると, button_click関数が呼び出される

```
6 def button_click(event):
7     btn = event.widget
8     txt = btn["text"]
9     tkm.showinfo(txt, f"[{txt}]ボタンが押されました")
```

← 左クリックされたウィジェットを表す
← 対象ウィジェットのtext属性の値



※bindのわかりやすいサイト :

<https://office54.net/python/tkinter/tkinter-bind-event>

練習問題：calc.py

少なくとも1問ずつ，ステージング，コミットするように

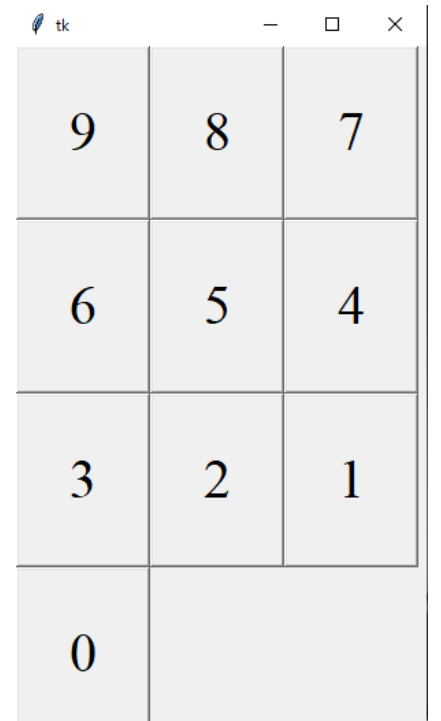
1. 300x500のウィンドウを作成せよ

2. 「0」～「9」の数字が書かれたボタンを作成し，図のようにせよ

- 幅：4／高さ：2
- フォント：("Times New Roman", 30)
- gridでrowとcolumnを設定

3. 10個のボタンに，クリックした際の反応を定義せよ

- 例：「1」のボタンがクリックされたら，
showinfoで「1のボタンがクリックされました」と表示させる



コーディング時に意識してみよう

- 読みやすさ：空行・空白，文中改行の入れ方
- コードの簡潔さ \leftrightarrow 冗長さ，長さ，一貫性
- 変数名，関数名，クラス名
- 一時変数の利用
- 全体の構造：クラス，関数を定義している
- ループの作り方：`for i in range`/`for x in list`
- ネストの深さ
- コメントの有無
- 修正容易性，拡張容易性

練習問題：calc.py

少なくとも1問ずつ、ステージング、コミットするように

4. テキスト入力欄を、10個のボタンの上部に追加せよ

- justify:right / width:10 / font:(Times New Roman, 40)
- 配置時にはgridにcolumnspanを設定するとよい

※適宜、ウィンドウ
サイズを変更すること

5. クリックした際の反応を以下のように変更せよ

- 4. のテキスト入力欄に、クリックしたボタンの数字を挿入する
- 挿入にはinsertメソッドを用いる
- 挿入位置は、tk.ENDを指定する
- 「tk.END」ではなく、「0」と指定した場合も試してみよう

練習問題：calc.py

少なくとも1問ずつ、ステージング、コミットするように

6. 空いている場所に「+」ボタンを追加し、クリックされたときの反応を関数として定義し、その関数をボタンにbindせよ
 - 「+」ボタンは数字ボタンと同じ挙動でもよい（まだ計算しない）
7. 空いている場所に「=」ボタンを追加し、クリックされたときの反応を関数として定義し、その関数をボタンにbindせよ
 - 4.のテキスト入力欄の数式を取得し、式を評価し、表示内容を削除し、計算結果を挿入する
 - 数式の取得にはgetメソッドを用いる
 - 式の評価にはeval関数を用いる
 - 表示内容の削除にはdeleteメソッドを用いる
 - 計算結果の挿入にはinsertメソッドを用いる

4限：演習課題

追加機能を実装する前に、現時点での最終版をpushすること

これ以降、指示があるまでpushしないこと

calc.pyを改良する

- 独自に，電卓の見た目を良くし，機能を充実させよう
- 追加機能の例：
普通の電卓では
 - 数字の配置は右図のようになっている
 - マウスオーバーした際にボタンの色が変わる
 - クリアボタンやオールクリアボタンで入力をリセットできる
 - 四則演算機能がある
 - 平方根の計算ができる
 - 二乗の計算ができる
 - %表示できる
 - 逆数の計算ができる
 - 数式としてふさわしくないボタン（例：「x+6-55÷÷÷4」）は押しても表示されない



別ブランチで作業しよう

3限に実装したコードを壊さないように、新たなブランチを作成し、新ブランチで作業する

- ブランチを切る：「`git branch ブランチ名`」
- ブランチを切り替える：「`git switch ブランチ名`」

例：「`git branch add_func`」 + 「`git switch add_func`」

- 別ブランチでも随時add/commitすること（pushはしない）
- すべての変更をcommitしたら「`git switch main`」でmainに戻る

ex02/README.mdを書く

- 本日の演習で追加実装した機能についての説明を追記してみよう例：

```
# 第2回
## 超高機能電卓 (ex02/calc.py)
### 追加機能
- 四則演算：+以外の四則演算をする
- クリア：entryに入力されている数字，数式の1文字をdeleteする
- オールクリア：entryに入力されている数字，数式の文字列全体をdeleteする

### ToDo (実装しようと思ったけど時間がなかった)
- [ ] 平方根ボタン:smile:
- [ ] べき乗ボタン

### メモ
- ブランチを作るコマンド：git branch ブランチ名
- ブランチを切り替えるコマンド：git switch ブランチ名
- mainブランチに戻るコマンド：git switch main
- ブランチを切り替える前に，すべての変更履歴をコミットした方がいい
```

【再掲】 マークダウン記法

書き方	範例
# 見出し	見出し
- 順序なしリスト	• 順序なしリスト
1. 順序付きリスト	1. 順序付きリスト
- [] チェックリスト	<input type="checkbox"/> チェックリスト
> 引用	引用
太字	太字
斜体	斜体
~~取消線~~	取消線

[link text](https:// "title")	リンク
![image alt](https:// "title")	画像
`コードブロック`	コードブロック
```javascript var i = 0; ```	<pre>1   var i = 0;</pre>
:smile:	

# ブランチをマージしよう

別ブランチでの開発がうまくいったら、mainブランチに戻り、別ブランチでの変更履歴をmainブランチに取り込む

- ブランチをマージする：「`git merge 別ブランチ名`」
- 例：「`git switch main`」 + 「`git merge add_func`」

マージが完了したら、別ブランチは不要なので削除する

- ブランチを削除する：「`git branch -d 別ブランチ名`」
- 例：「`git branch -d add_func`」

# 5限：グループワーク

追加機能を実装後， mainブランチにmergeしたら， pushする

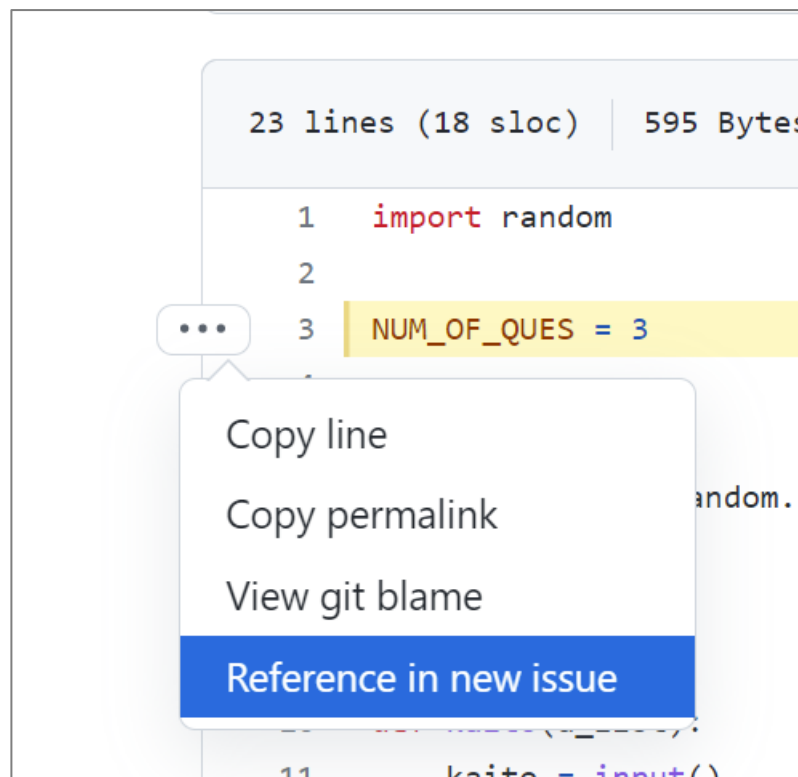
# グループを作り，コードを読む

- Githubで共有されたREADME.mdとコードを読む
- ZOOMのBORでの画面共有により，追加機能をデモする
- コードについて議論する
- 説明者以外の4人のうち最低2人は，コメントをつける
  - ※必ず，全員が2人以上からコメントを受けること
  - ※必ず，全員が2人以上にコメントを付けること
- Issuesでコメントを受けて，
  - 修正すべきなら修正する
  - 修正すべきでないなら，修正しない理由を返信する
- コード修正のコメントがない場合は，TASAからのコメントを受ける

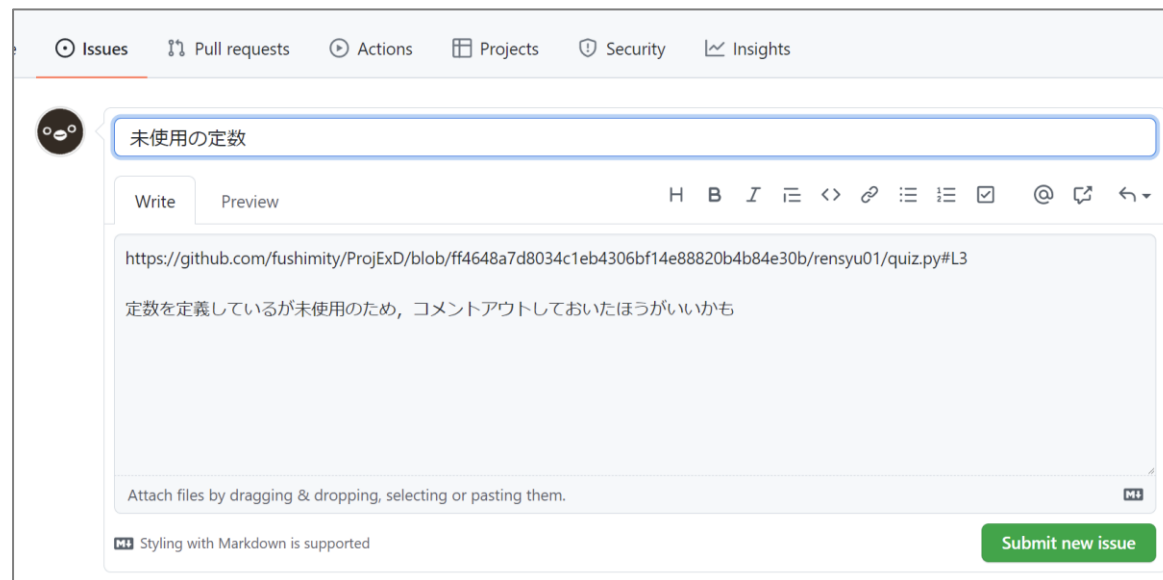
# 【再掲】 Issueでコメントを書いてみる

- 当該コードを読み，必要に応じてコメントを送る

①コメントしたい行にカーソルを当て，「...」→「Reference in new issue」

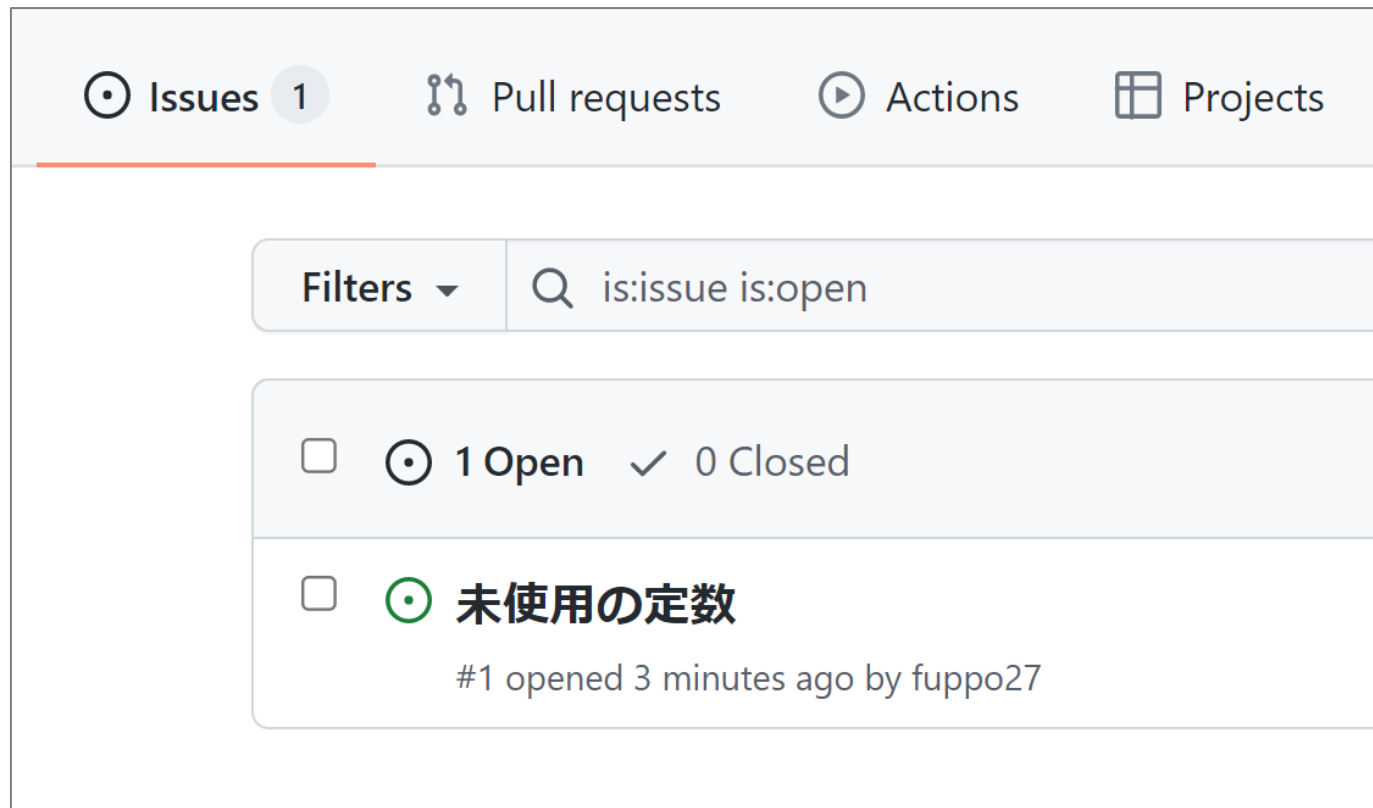


②コメントを書き，「Submit new issue」



# 【再掲】 Issueが届く

- Issuesタブから, Issueのコメントを読み, 対応する



## 【再掲】 Issueコメントを受けてコードを修正する

1. Issue#1のコメントを受けてコードを修正する
2. 修正が終わったらステージングする「`git add ex01/quiz.py`」
3. ステージングされた内容をコミットする  
※重要：コミットコメントに、Issue番号を付けること

「`git commit -m "コメント #1"`」

要半角スペース

これにより、  
Issueコメントと対応コミットがGithub上で紐づけられる

4. リモートリポジトリにプッシュする「`git push origin main`」



# 提出物

学籍番号は, 半角・大文字で

- ファイル名 : C0A21XXX_kadai02_.pdf

- 内容 :

自分のアカウント名

- READMEの最終版 ( rendered blob )

- <https://github.com/fushimity/ProjExD/blob/main/ex02/README.md>

- コード差分表示 ( 3限<基本機能> vs 4限<追加機能> )

- <https://github.com/fushimity/ProjExD/commit/49e9a69ffe155ead17>

- コード差分表示 ( 4限<追加機能> vs 5限<コメント修正> )

- <https://github.com/fushimity/ProjExD/commit/46118f25c12d471ca3>

- Issues一覧

- <https://github.com/fushimity/ProjExD/issues>

該当コミットID

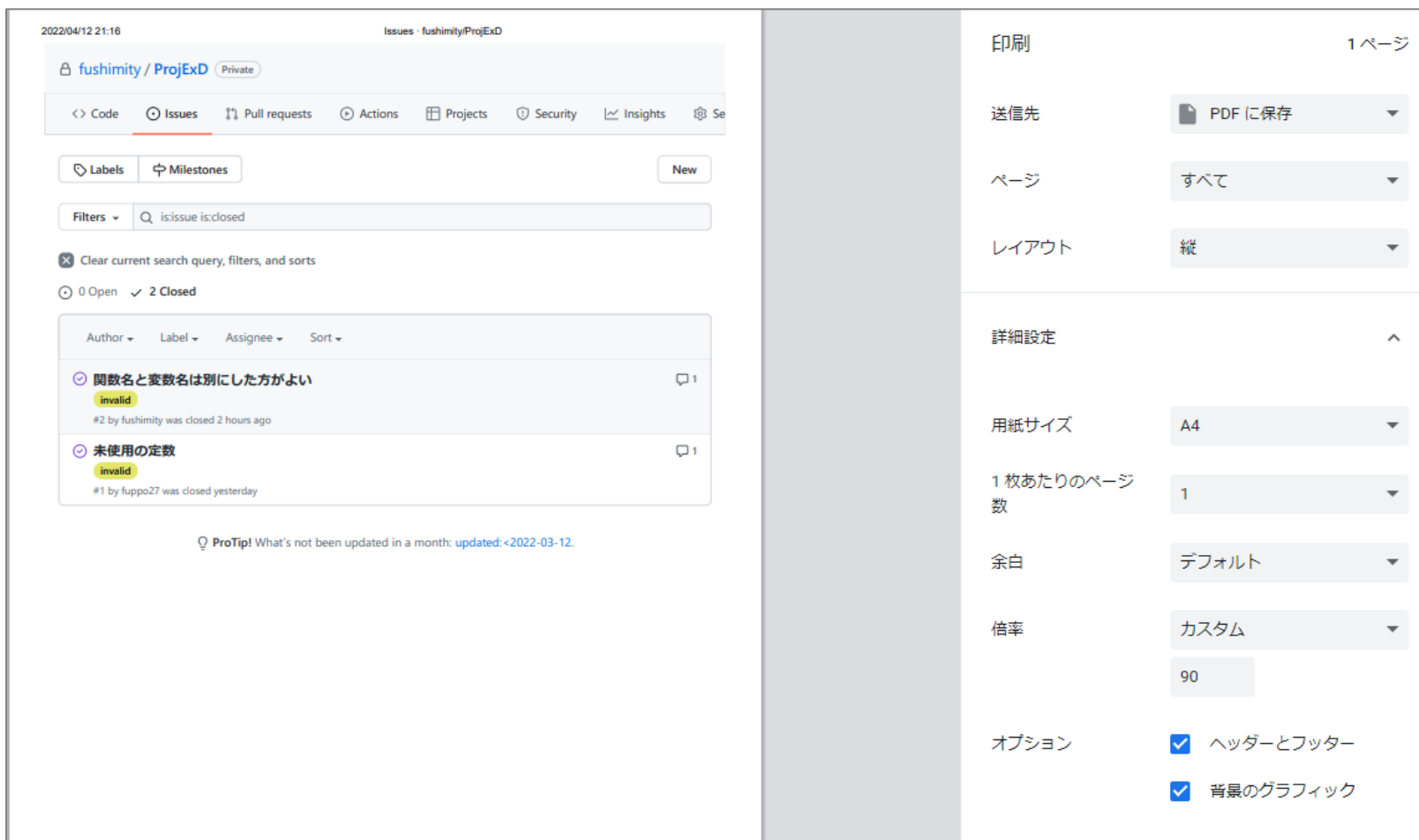
# チェック項目

時間内 : 100%  
当日中 : 80%  
今週中 : 50%  
次回まで : 10%

- READMEの説明は十分か [0 -- 2]
  - わからない [0], まあわかる [1], よくわかる [2]
- 複数の機能を追加しているか [0 or 2 or 4]
  - 追加機能なし [0], 講義資料の追加機能 [2], 独特の追加機能 [4]
- Issueで複数人のコメントを受けているか [0 -- 2]
  - コメント人数(TASAを除く)に基づく。ただし2人以上は [2]
- コメントに対して、適切に反映、返信しているか [0 -- 2]
  - 対応コメント数に基づく。ただし2コメント以上は [2]
- calc.pyはP.19の点を考慮しているか [0 -- 5]

# 【再掲】 ChromeでPDFとして保存する方法

1. 該当ページを表示させた状態で「Ctrl+P」
2. 以下のように設定し、「保存」をクリックする



←送信先：PDFに保存

←ページ：すべて

←レイアウト：縦

←用紙サイズ：A4

←余白：デフォルト

←倍率：90

←両方チェック

# 【再掲】 各PDFを単一ファイルにする方法

1. ChromeでPDFとして保存する
  2. 以下のURLから各PDFをマージする
  3. ファイル名を「C0A21XXX_kadai02.pdf」として保存する
- オンラインでPDFをマージするサービスの例：
    - [https://www.ilovepdf.com/ja/merge_pdf](https://www.ilovepdf.com/ja/merge_pdf)
    - <https://chrome.google.com/webstore/detail/merge-pdf/ehbfcoenegfhpnnmkoaimmmlhikfccli/related?hl=ja>

# 【再掲】 提出，チェックの流れ

1. 受講生：提出物ができたらMoodleにアップロードする
2. 受講生：課題チェック依頼のスプレッドシートにて，待ちキューが少ないTASAの列に学籍番号を入力する
3. TASA：Moodleにアップされた課題をチェックする
4. TASA：チェックが済んだら，
  - Moodleに点数を入力する
  - スプレッドシートの学籍番号を赤字に変更する【チェック完了の合図】
5. 受講生：自分の学籍番号が赤字になったら帰る
  - チェックを待たず帰ることも可能だが，提出物に不備があっても修正できない