論文

GUI の自動操作とスクリーンショットに基づく GUI 採点支援機能を有した Python プログラミング実習用システムの開発

毛利　元昭（愛知大学経営学部）

要旨

プログラミング実習系授業の実施において，担当教員にかかる負担の軽減や学習者へのフィードバックの早期化は重要である．既存のプログラミング実習システムの多くは CLI アプリケーション用の自動採点機能を有しているが，GUI アプリケーション用は実装例が少なく，利用可能な状況も限られている．そこで筆者は，GUI 自動操作とスクリーンショットに基づく GUI 採点支援機能を有した Python プログラミング実習用システムを開発した．最終的な採点は人が行う必要があるものの，手元の環境での実行を減らせるため，採点時間が短縮され，負担軽減やフィードバックの早期化を図ることができた．

キーワード： プログラミング学習，採点支援，Python，GUI，PHP，シェルスクリプト

1. はじめに

プログラミングの実習系授業の実施において，担当教員にかかる負担は大きい．資料・教材作成と講義だけでなく，課題の細やかな出題，学習者の課題進捗状況の把握，採点とフィードバック，質問やトラブルへの対応などが求められる．また，Try and Error を繰り返すプログラミング学習では，合否のフィードバックはできるだけ早期であることが望まれる．情報工学や情報科学を専攻していない初心者 50〜60 人規模の複数クラスを単独で担当するような場合，授業を正常に実施するには何らかの効率化・自動化が必要と言える．

このような背景から，これまでに多くのプログラミング実習用システムが開発され，広く利用されている．日本人の利用が多いものとしては，Aizu Online Judge[注 1)]，Progate[注 2)]，paiza[注 3)] などがあり，世界的には AtCoder[注 4)] が有名である．しかしこれらのほとんどは，ソースコードあるいはコマンドラインインターフェース（CLI）上でのテキスト入出力に基づいた自動採点機能を有するにとどまっている．その理由は下記のように考えられる．

1. 任意のプログラミング言語で記述された GUI を，ウェブブラウザ上に構成できない

2. 初心者である学習者に，出題者の意図通りのテスト（GUI の操作）を，正確に行わせることが困難
3. 結果の判定・評価が困難

箇条書きの 1 について詳細を述べる．まず，プログラミング学習をウェブサービスとして提供する場合，学習者側の環境はウェブブラウザが前提となることが多い．学習者がウェブブラウザを介してコードを提出すると，その実行結果や評価結果がウェブブラウザ上に表示される形である．しかし，ウェブブラウザ上で GUI を構成可能なプログラミング言語は，現在では実質的に JavaScript に限定される．その JavaScript も，GUI に対しては実際には DOM[注 5)] を介して HTML 文書内の要素にアクセスする形である．つまり，任意のプログラミング言語で記述された GUI をブラウザ上に構成するには，GUI の各パーツを HTML 要素に変換することが必要となり，それらを扱うにはプログラム全体を JavaScript に変換することが必要となる．これは，技術的には可能かもしれないが，そのようなツールの開発コストやテストにかかるリソースについて勘案すると，授業・サービスを提供していく上で現実的でない．実際に，筆者が担当する授業で教えている Python に限定しても，標準の GUI ライブラリである Tkinter を HTML および JavaScript に変換するツールは，2025 年現在で見つけられなかった．

ここで，ウェブブラウザ上に GUI そのものを構成せず，画像として表示，あるいはレイアウト情報として表示することは，さほど難しくない．実際に，筆者が以前に開発したプログラミング実習用システム[1)]でも，GUI の起動時のウィンドウをスクリーンショット（SS）し，実行結果として表示させていた．また，立石らが開発したシステム[2)]では，Java ソースファイルから GUI のレイアウト情報を XML 形式で出力させていた．しかし，GUI が構成されているわけではないため，これらの方法ではウェブブラウザ上での GUI のテストを行うことができない．

箇条書きの 2 について詳細を述べる．GUI を備えたアプリケーション（以下，アプリ）の実行・テストには，GUI の操作や GUI 上への表示が伴う．GUI が想定するマウスやキーボードを介した操作を，ウェブサーバ側からウェブブラウザ上で自動で行わせることは，セキュリティ上の理由からできない．学習者による実際の操作，あるいはローカル環境での何らかのユーザ操作自動化ツールが必要となる．

ここで，学習者自身の手でテストを行わせることは，プログラミングへの理解を深めることには繋がるものの，課題ごとに異なるテストを正確に行わせる必要があり，初心者を対象とする授業・サー

ビスを提供していく上で現実的でない．大学での成績評価には公平性と公正性が求められることからも，採点結果を自己申告させる形は避けるべきであろう．

ユーザ操作自動化ツールを用いる場合も，その環境の構築・設定を学習者に求めることになり，あるいは学習者の多様な環境に応じて対処する必要があり，やはり初心者に対しては現実的でない．例えば，Selenium[注 6)]と呼ばれる操作自動化ツールは，その操作内容を Python で記述することができるが，操作対象はウェブブラウザに限定されている．Ranorex[注 7)]と呼ばれるテスト自動化ツールは，操作対象を問わないが，利用環境が Windows に限定されている上に，有償である．xdotool[注 8)]と呼ばれる操作自動化ツールは，操作対象を問わないが，利用環境が Linux または macOS に限定されている上に，CLI 上で実行する必要がある．PyAutoGUI[注 9)]と呼ばれるライブラリは，Python プログラム内に GUI 操作を記述できるが，「指定タイトルのウィンドウを見つける」などのウィンドウ操作が環境依存であり，環境に応じて他のライブラリと組み合わせる必要がある．

箇条書きの 3 について詳細を述べる．何らかの方法で，ユーザ側あるいはサーバ側で GUI の操作を自動的に行えるようになったとする．しかし，その結果となる GUI 上への表示をどのように判定・評価すべきかについては，未だ決定的な方法が確立されていない．

前述の立石らの GUI アプリ自動採点システム[2)]は，解析した GUI のレイアウト情報を利用してサーバ側で GUI の自動操作を行っており，複数のテスト結果に基づいて評価を行っていた．また，テストケースを GUI の操作ベースで生成するツールも付属していた．しかし，対象となる言語は Java に，GUI ライブラリは JavaFX に限定されており，Python を教えている筆者の授業では利用できない．水谷は，大規模言語モデルの一種であり，画像の言語化を得意とする Vision Language Model (VLM) を用いた GUI アプリ自動採点システム[3)]を提案した．しかし，計算リソースへの要求が高い VLM をオンプレミスで構築する必要があり，計算量・計算時間が膨大になる見込みで，かつ，システムとしても検証段階で成熟していない．

これらの理由から，GUI アプリの自動採点は現状では困難であり，GUI アプリの最終的な採点は未だ教員が行う必要があると判断した．同時に，筆者はプログラミング実習用システム[注 10)]に GUI アプリの採点支援としての機能実装を行うこととした．本稿では，開発したシステムの設計・構成を説明するとともに，採点時間を比較することで本システムの有用性を示す．

2. 採点者にかかる負担とその軽減

プログラミング実習用システムに自動採点が期待される理由は，採点する教員にかかる負担の軽減にある．それでは，採点者はどのような状況に負担を感じるのか．筆者は次のように考えた．

1. 採点を必要とする対象が多い
2. 採点に必要なステップ数が多い
3. 正解の状態との比較がしにくい
4. コメントでのフィードバック

それぞれへの対策について，方針を述べる．

2.1. 採点を必要とする対象が多い

そもそもの学習者の数や課題の数を減らすことは考えるべきではないため，手動採点の必要性を減らすことを考える．学習者は，課題が合格の要件を満たしているという自信が無い場合でも提出することがある．特に初心者はテストを自主的に十分行おうとせず，実質的にテストを採点者に委ねるために提出することが多い．逆に言えば，提出したプログラムが合格の要件を満たしていないことが分かれば，学習者は修正作業に取り掛かることができる．ここで，明らかに要件を満たしていない場合にも採点の実施を多数要求されることは，採点者にとって大きな負担となる．よって少なくとも，自動判定可能な要件については自動判定を実施し，満たしていない場合は「不合格」とする形（足切り）が良いと言える．

プログラムが合格の要件を満たしているかどうかを判定するには，プログラムを解析する必要がある．本システムの前身[1)]では Java をビルドした JAR 形式で提出させており，その JAR は中間言語へのコンパイルと最適化が行われたものであった．そのため，JAR を javap [注 11)] コマンドで解析した結果は，穴埋め形式のコーディングであれば学習者ごとの揺らぎが少なく，自動採点に活用できた．一方，本システムで前提とする Python は，表面上，プログラムをソースコードの形で保持しつつ実行する．そのため，ソースコードそのものを提出させる形が最も自然であるが，そのままでは学習者ごとの揺らぎが大きい．ソースコードを自動解析するにあたって，前処理として表記のある程度の画一化処理が必要である．

ここで，Python には isort [注 12)]，autoflake [注 13)]，black [注 14)] などの自動整形ツールが開発されている．自動整形ツールは，チーム開発などの際にソースコードの表記の画一化を目的として導入されるものであるため，今回の目的にも合致する．よって，自動整形ツールを適用した後にソースコードを解析し，合格の要件を満たしていない場合は即座に「不合格」とすればよい．

2.2. 採点に必要なステップ数が多い

前述の足切りによって手動採点の対象が減った場合でも，採点 1 件当たりに必要なステップ数が多いと，採点者にとって大きな負担となる．ウェブブラウザ上で実行できない GUI アプリの手動採点にかかる作業は，一般的に次の通りであろう．

1. 採点を必要とする対象を探す
2. 答案をダウンロードする
3. プログラムを実行・操作する
4. 正解の状態と比較する
5. 評価やコメントを登録する

箇条書きの 4 と 5 については，完全な自動採点を目指さない以上，手動での作業を無くすことはできない．これらの負担の軽減については，第 2.3節および第 2.4節で述べる．ここでは，箇条書きの 1〜3 について，以降に詳細を述べる．

2.2.1. 採点を必要とする対象を探す

手動採点の画面の作り方にもよるが，採点を必要とする対象を探すのに手間取ることは，採点者にとって負担である．筆者の場合，GUI アプリ作成の科目には毎年 30 人程度の履修者がおり，授業中のガイドに従ってほぼ一斉に課題が提出される場合もあれば，任意のタイミングの場合もある．いずれにせよ，そのままでは採点を必要とするものと不要とするものが入り乱れている状況となる．ここで，「探す」をステップ数に含めなくてよい状態とは，例えば任意の課題の答案一覧をページ表示した際，採点を必要とする答案が分かりやすく表示されている状態である．つまり，何らかの強調機能やフィルタ機能が備わっていることが望ましいと言える．

2.2.2. 答案をダウンロードする

ウェブブラウザ上で実行できないため，手動での動作確認をするには採点者のローカル環境で実行する必要がある．最近のウェブブラウザは，セキュリティの関係から，プログラムのダウンロードおよび実行の際には拒否を前提とする確認ダイアログが表示される．セキュリティの設定を変更することで確認不要で許可とすることもできるが，学内備品の PC 上での設定変更は問題となりやすい．加えて，アプリの実行に必要なファイルがプログラム本体の他にも存在する場合，それらも正しく配置しておくことが必要であり，実質的なステップ数が増える．これらをステップ数に含めないようにするには，プログラムをダウンロードをすることなく，ウェブブラウザ上で目視で採点できることが望ましいと言える．

2.2.3. プログラムを実行・操作する

GUI アプリの動作確認をするには，採点者が確認事項とそのための操作について正しく理解し実施する必要がある．現状で，筆者の科目では筆者のみが採点者で，同じ課題の答案を連続して採点していくとしても，数をこなすことはやはり負担である．これをステップ数に含めないようにするには，確認事項に対応する操作が自動でなされた様子を確認できることが望ましいと言える．つまり，GUI の操作自動化ツールを導入する必要がある．

2.3. 正解の状態との比較がしにくい

GUI アプリの動作確認をするには，確認事項に対応する操作が自動でなされた様子を確認するとしても，正解の状態と比較することが必要である．筆者は出題者でもあるため，課題ごとに自身で設けた採点基準に基づいて採点を実施しているが，その負担は大きい．採点者にかかる負担を減らすには，正解例を操作した際の様子が，比較しやすい形で提供されていることが望ましいと言える．

ここで，GUI の自動操作ツールを利用したとして，それによって正しく操作できるかは事前の確認が必須である．その様子を動画あるいはそれに準ずるものとして記録しておき，学習者の提出したプログラムが同様に操作された様子と並べて表示させればよい．ただし，動画だとして，その再生タイミングにズレが生じていると，比較が困難となる．再生タイミングを同期させる仕組みを組み込む必要がある．

2.4. コメントでのフィードバック

採点においては，特に「不合格」とした際に，学習者にその理由等の説明や修正のための助言を与えたい場合がある．筆者がこれまで受け持った学習者の多くは，手元で実行してもエラーメッセージを細かく読むことを敬遠していた．直接的な修正方法を述べたものではないことに加え，専門用語を含んだ英文のため，「読んでも分からない」との印象があったためである．その結果，正解例が示されるまで取り組まない者や，質問対応で正解例ばかりを求める者も多々現れた．その状況は授業進捗の遅延と教員の負担を生じさせるため，削減が望まれる．

ここで，前述のように自動判定可能な要件については自動判定を実施する形であれば，その判定理由は明らかであり，修正のための助言も多くは定型的である．つまり，ある程度であればコメントでのフィードバックは自動化可能であり，採点者にかかる負担を軽減できると言える．

3. システム設計

3.1. システム要求

GUI 採点支援機能を有したプログラミング実習用システムの開発にあたり考慮したシステム要求は，次の通りである．

1. 学習者がウェブブラウザを介してアクセスでき，課題の表示や提出，評価の確認も可能
2. 60 人程度がほぼ同時にアクセスしてもシステムダウンが発生しない
3. 利用にはログインを必要とし，テスト用ユーザを除いて学内の認証サービスにて認証
4. プログラム（および実行に必要なファイル群）が提出された場合，プログラムの解析および実行を自動で行い，その結果をファイルとして保持
5. 課題の提出から採点結果等の表示までが 20 秒以内
6. GUI を含め，課題の実行結果を課題用ページにて並べて表示
7. 管理者の画面から，提出されたファイルや答案の確認，採点の修正や手動採点，コメントが可能
8. 管理者の画面から，クラスごと・授業の回ごとに課題の公開・提出期間等の設定が可能

これらの要求が満たせるように，オンプレミスにて環境構築およびシステム設計を行った．現行の環境は表 1の通りである．

表 1: システムの環境

本体	PC/AT 互換機
OS	Lubuntu 24.04 64bit
ウェブサーバ	Apache 2.4
サーバ側処理系	PHP 8.3, Bash 5.2, Python 3.12
GUI 自動操作	xdotool 3
画面・画像処理	scrot 1.10, ImageMagick 6.9, APNG Assembler 2.91
ブラウザ側処理	JavaScript, jQuery 3.7
データベース	MariaDB 15.1

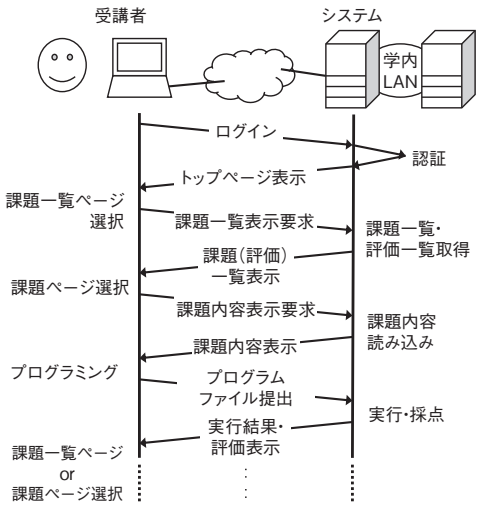


図 1: 学習者によるシステム利用の流れ

学習者によるシステム利用の流れは図 1の通りであり，本システムの前身と同様であるため，詳細な説明を割愛する．

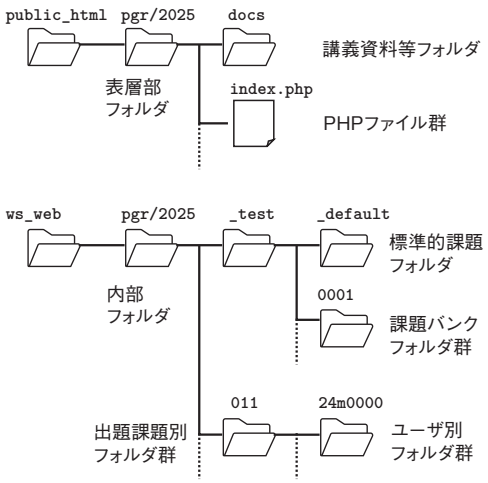


図 2: システムのフォルダ構成

3.2. 課題バンクフォルダ群

フォルダ構成は図 2の通りであり，基本的に本システムの前身と同様であるため，ファイル群に変更点のある課題バンクフォルダ群についてのみ説明する．

「_test」フォルダ内の課題バンクフォルダ群においては，各課題の識別子（数字 4 桁）をフォルダ名として課題ごとにフォルダを用意している．課題ごとのフォルダには，主に下記のファイルが指定の名前で収められている．

- setting.txt
  DB への登録用設定ファイル
- direction.txt
  課題の説明（専用）
- figure.png
  説明の補足用の画像
- code.txt
  ソースコード（提示用）
- answer.txt
  正答例
- test_cmd.sh
  自動採点スクリプト（専用）
- input*.dat（*は任意の数値）
  自動実行時の入力
- output*.dat（*は input に対応）
  自動実行時の想定出力（比較用）
- xdotool.txt
  GUI 自動操作内容（xdotool 準拠）
- ss_win.png
  GUI 実行時の想定ウィンドウ SS（アニメーション）
- output_pc2.txt
  クラスおよび関数・メソッドの情報（比較用）
- その他，配布ファイルなど

本システムの前身からの変更点は，「xdotool.txt」および「ss_win.png」が追加されていることである．これらの利用については，第 5章で詳細を述べる．

また，課題ごとのフォルダとは別に，標準的な課題用の「_default」フォルダを用意してある．そこには課題の説明（direction*.txt）と採点スクリプト（test_cmd*.sh）を課題のタイプ別（*はタイプの番号に対応）で収めてあるほか，共用のプログラムなどを収めてある．課題ごとのフォルダ内に各ファイルが存在しなければ，これらが利用される．課題

のタイプは次の 8 通りとする.

1. プログラムファイルを受け取るのみ
2. CLI アプリのファイルを受け取って自動採点
3. GUI アプリのファイルを受け取って自動実行・自動操作
4. テキストデータを受け取るのみ
5. テキストデータを行ごとに自動採点
6. PNG ファイルを受け取るのみ
7. ドキュメントを受け取るのみ
8. ZIP ファイルを受け取るのみ

共用のプログラムは次の 2 つである.

- make_pc2.py
  Python コードからクラスおよび関数・メソッドの情報を抜粋
- test_xdo.sh
  xdotool.txt に従って GUI を自動操作しながらウィンドウ SS のアニメーションを生成

これらを含む，採点スクリプトおよび GUI の自動操作については第5章で詳細を述べる.

3.3. データベース構成

データベース構成も，基本的に本システムの前身と同様であるため，説明を割愛する.

4. PHP ファイル群

PHP ファイル群は，大きく分けて次の 4 つに分けられる.

1. ユーザ情報および認証関連
2. 課題および評価の提示関連
3. 課題の提出および採点関連
4. 管理関連

1 は論旨に関わらないため，説明を割愛する．2〜4 は，主に本システムの前身からの変更が大きな点について述べる.

4.1. 課題および評価の提示関連

該当するファイルは次の通りである.

- show_eu.php
  課題（評価）一覧ページ
- show_p.php
  課題内容ページ
- show_p9.php
  授業への意見等を入力するページ
- disp_png.php
  システムの内側に保存された PNG 画像の表示
- dl_file_d.php
  課題の配布ファイルのダウンロード
- dl_file_u.php
  提出ファイルのダウンロード

この中で論旨と関係の深い show_p.php について，詳細を説明する．

show_p.php は，課題の詳細な内容が表示されるとともに，答案の提出を受け付けるページである．その一例を図 3に示す．課題番号はクラスと授業回（2 桁）と課題の順番からなるが，学習者に作成させる課題用のフォルダ名も兼ねている．そのため，課題の説明文中のプロジェクト名が課題番号になるよう，表示時に置換している．メインの表には，タイトルに続き，説明，コード，図，配布物，提出，判定，コメント，正解例の欄が並ぶが，コード以降の欄はそれぞれ不要な場合には表示されない．判定欄には，下記のマークのいずれかが表示される．

- ”×”：　未提出
- ”▲”：　不合格
- ”確 (遅)”：　確認中（遅刻提出）
- ”●”：　合格（遅刻提出）
- ”確認中”：　確認中・未評価
- ”◯”：　合格
- ”◎”：　合格（加点対象）

メインの表の下には，前回提出したプログラムの実行結果と正解プログラムの実行結果を横に並べて表示する．テキストが出力されるアプリの場合は，相違点の背景をオレンジ色にすることで，プログラムの誤りを検討しやすくしてある．提出物が GUI アプリの場合は，自動実行

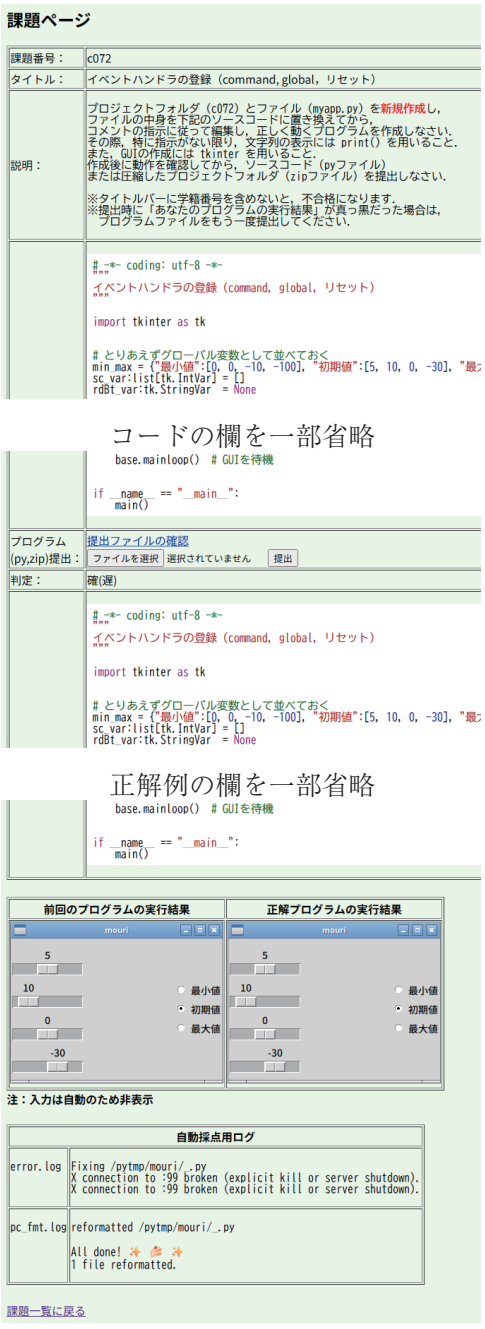


図 3: GUI を含む課題の内容ページ（show_p.php）の例

時のアニメーション（提出時の自動操作を指定していない場合は起動時のSS）を表示する．ここで，HTMLドキュメント中のimgタグのloading=”lazy”オプションによって，実質的にアニメーションの開始タイミングである表示タイミングを同時にすることで，動作状況の正解例との比較がしやすくしてある．課題の提出時だけでなく，事後にも「前回の結果」を確認・比較できる状況を用意することで，学習者個人へのフィードバックがなされるとともに，自主的な修正が促される．

これらの他，採点スクリプト実行時のログファイル群（*.log）の中身を列記する．本来の用途はデバッグだが，第2.4節で述べた自動的なコメントとしての，評価結果に対する学習者への説明にも活用している．よくある誤りへの指摘などを，採点スクリプトを介してログファイルに出力させることで，実質的に学習者へのコメントを自動化する形である．これにより，学習者個人へのフィードバックを早期化できる．

なお，評価結果が合格（加点対象）の際は，花火のアニメーションを表示する．些細な報酬ではあるが，学習者のモチベーション向上に役立っている．

4.2. 課題の提出および採点関連

該当するファイルは次の通りである．

- ul_code.php
  プログラムファイルの提出，採点スクリプトの実行，結果表示
- ul_text.php
  テキストエリアのデータの投稿，採点スクリプトの実行，結果表示
- ul_file.php
  その他の指定ファイルの提出
- submit_p9.php
  授業への意見等を提出

この中で論旨と関係が深いul_code.phpについて，詳細を説明する．

ul_code.phpは，課題タイプが1〜3の場合に，show_p.phpからプログラムファイルをPOSTすると呼び出される．出題課題別・ユーザ別フォルダへのファイルの格納，採点スクリプトの実行とその結果の表示を行う．

具体的には，まず，提出されたファイルの拡張子が[py, pyw, txt]の場合は「myapp.py」に，[pyz[注15)], zip]の場合は「a.zip」にリネームし，出題課題別・ユーザ別フォルダへ格納する．圧縮ファイルである[pyz, zip]を許容する理由は，アプリの実行に必要なファイル群をまとめて提出させるためである．ここで，圧縮ファイルの場合は展開も行うが，展開後にmyapp.pyが存在しない場合はエラー表示を行う．なお，この一連の工程は，採点スクリプトを実行するにあたり，プログラムファイル名を固定するために行

うものである．

続いて，実行フォルダを出題課題別・ユーザ別フォルダに移し，timeout コマンド注 16)とともに採点スクリプトを実行する．これにより，無限ループなど実行時のトラブルが発生してもシステムが機能不全に陥らない．GUI が含まれる課題タイプ 3 の場合は描画領域が必要なため，別途 xvfb-run コマンド注 17)とともに自動操作スクリプトを実行する．このとき，ブラウザの応答がタイムアウトすることを防ぐため，プロセスを分離させる．

採点スクリプトは，リダイレクトを用いて途中の標準出力や標準エラー出力を全てそれぞれ指定のファイルに吐き出し，最終的な評価値を表す 1 文字（不合格：”F”，確認中：”K”，合格：”G”，合格（加点対象）：”S”）のみを標準出力する形をとる．この評価値を PHP 側で受け取り，それに基づいてデータベースへの評価の登録と，ユーザへ示す評価結果ページを作成・表示する．

評価結果ページには，課題内容ページ「show_p.php」と同様に，GUI の画像を含め，提出されたプログラムの実行結果と正解プログラムの実行結果を並べて表示する．採点スクリプト実行時のログファイル群や加点対象への花火アニメーションも同様である．ただし，ページロード時には GUI の自動操作が未完了の場合があり，その場合は起動時の SS（静止画）が表示される．学習者が課題内容ページに戻る頃にはアニメーションが完成していることが多いため，この対処で問題は生じにくい．

4.3. 管理関連

該当するファイルは次の通りである．

- admin.php
  管理用の機能へのアクセスページ
- show_ls.php
  授業情報の一覧ページ
- edit_l.php, update_l.php
  授業ごとの設定ページと更新
- edit_el.php, update_es.php
  授業ごとの評価の管理ページと更新
- show_ap.php
  課題ごとの答案の一覧ページ
- read_p9.php
  授業ごとの意見の確認ページ
- update_ps.php
  課題バンク情報の DB への登録
- show_es.php
  数値化した評価の一覧ページ
- show_swd.php
  内部フォルダの表示
- achive_swd.php, dl_file_s.php
  内部フォルダの圧縮とダウンロード
- edit_us.php, update_us.php
  ユーザ情報の管理ページと更新
- remove.php

ユーザ情報等の削除

この中で論旨と関係の深い show_ap.php について，詳細を説明する．

show_ap.php は，同一課題における同一クラスの全学習者の答案・評価を表示することに加え，各評価の登録・編集が可能なページである．その一例を図 4に示す．横方向には更新用チェックボックス，学籍番号（学習者の ID），評価，ファイル，コメントの欄が並んでおり，縦方向に各学習者の答案情報が並んでいる．表の 2 行目にあるドロップダウンリストは，第 2.2.1項で述べた，答案を評価でフィルタリングする機能である．これにより，「確認中」のみを表示させるなどして，効率的に採点できる．

ファイルの欄は，標準では提出されたファイルへのリンクやテキストの答案を表示し，課題別・ユーザ別フォルダ内の画像ファイル「a.png」「ss_win.png」が存在する場合は評価が「確認中」の場合にそれを表示する．これも，手動採点が必要な対象を優先的に確認するための仕様である．ただし，図 4中であれば表の 2 行 4 列目のオプションをクリックすることで，全員分の画像や，課題別・ユーザ別フォルダの任意のテキストファイルの中身を表示できる．具体的には，全員の画像を閲覧したい場合は「全」を，ソースコードを閲覧したい場合は「py」を，出力を閲覧したい場合は「dat」項の数字を，採点スクリプトによって生成される「output_pc*.txt」を閲覧したい場合は「pc」項の数字を，エラーログを閲覧したい場合は「err」をクリックすればよい．これにより，手動のダウンロードと実行を経ずとも自動実行された結果から教員が判定を下せるケースが生じるため，第 2.2.2項で述べたように採点者の負担軽減とフィードバックの早期化に繋がる．

課題バンクフォルダ内に，GUI 実行時の想定ウィンドウ SS（アニメーション）である ss_win.png が存在する場合，学習者のプログラム由来の ss_win.png の右側に並べて表示させている．課題ページと同様に，HTML ドキュメント中の img タグの loading=”lazy”オプションによって，実質的にアニメーションの開始タイミングである表示タイミングを同時にすることで，第 2.3節で述べたように動作状況の比較がしやすくしてある．ここで，想定画像は同じ課題で本来すべて同一となり，キャッシュによってそのアニメーションのタイミングは同一画像で一致してしまう．隣接する学習者由来の ss_win.png とタイミングが異なると，その両者の比較，ひいてはウェブブラウザ上での教員の目視による採点が困難になる．そこで，img タグの src の指定時に学籍番号をパラメータに加えることで，キャッシュの使いまわしを回避させ，アニメーションの開始タイミングが隣接する学習者由来の ss_win.png と同期させて

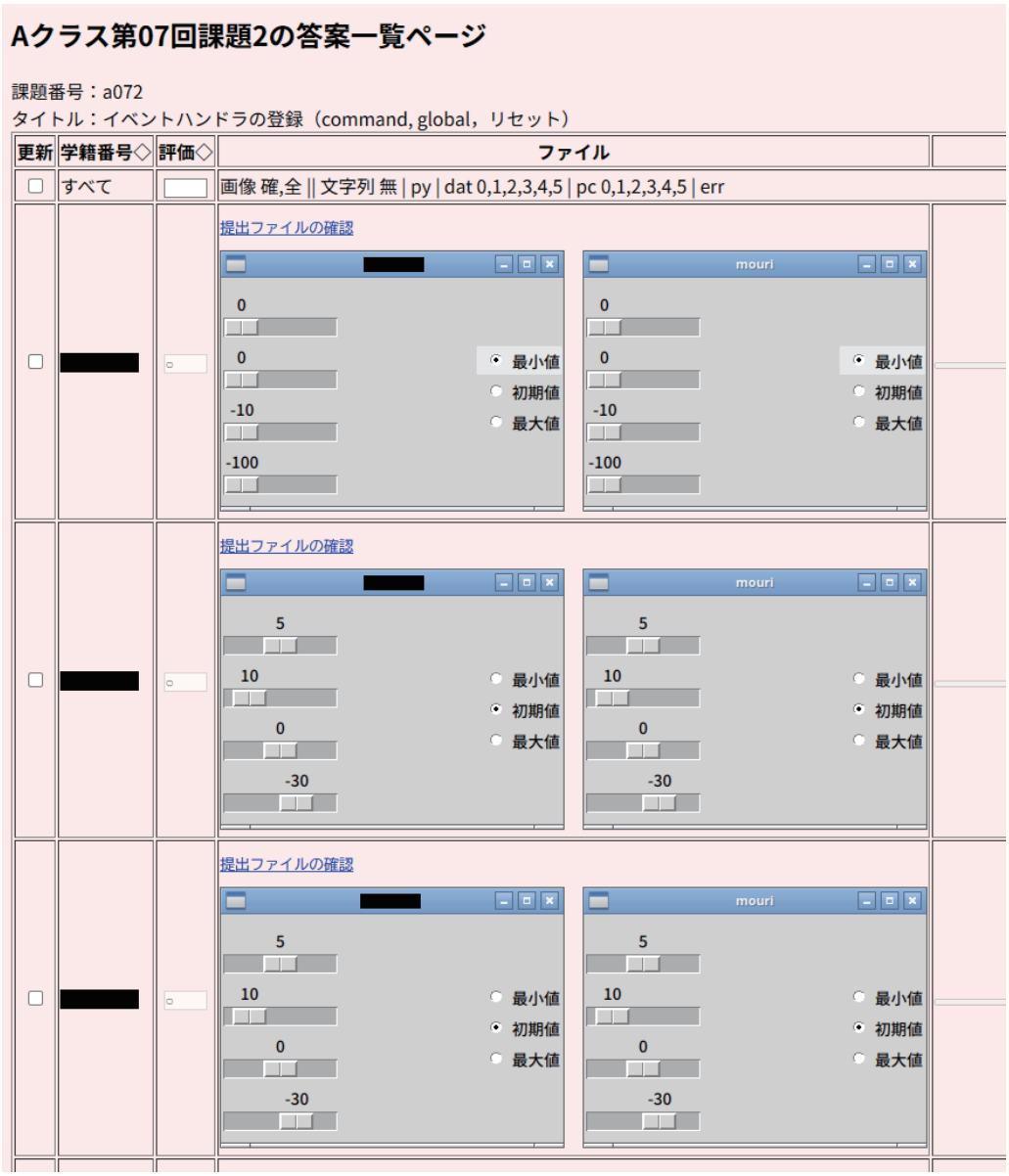


図 4: 答案一覧ページ（show_ap.php）の例

ある．実際に，図 4でも，横並びは画像が一致しつつも縦並びではアニメーションのタイミングが異なっていることが分かる．

コメント欄には，学習者個人へのコメントを書きこむことができる．筆者は主に，手動採点で評価を「不合格」にした場合に，その理由および助言を伝えるために利用している．

ページの最下段には，評価やコメント

の登録のためのボタンが配置してある．一覧の形で操作する都合上，評価やコメントは課題ごとにまとめての入力となる．

5. 自動採点・GUI 採点支援機能

5.1. 要求

自動採点機能および GUI 採点支援機能を実装するにあたって要求としたことは次の通りである．

1. Linux で利用できる無償のソフトウェア群・コマンド群で構成・構築
2. PHP から呼び出し，その採点結果を PHP から参照可能
3. Python プログラムを整形・解析し，その結果を採点に利用可能
4. Python プログラムを実行し，主に標準入力からデータを与え，標準出力は指定のファイルに保存
5. GUI アプリを自動操作し，その様子を APNG[注 18)]で記録
6. テキストの自動採点は別解に対応

それぞれの理由を簡潔に説明する．まず 1 については，筆者がシステム用の特別な予算を有していないためである．また，サーバの OS が Linux であることは前提のため，その上で利用可能な形で全てを構成・構築する必要がある．

2 については，表層部を PHP で構成することが前提のためである．PHP からの呼び出し方法や参照方法の自由度は高いため，容易な要求と言える．

3 については，プログラムの構造も採点対象に含めるためである．第 2.1節で述べたように，要件を満たしていない場合は「不合格」を学習者に伝えられた方がよい．また，解析に先駆けてソースコードを整形し，提出物とは別に保持しておく必要がある．

4 については，入出力に着目した採点のためである．処理用のデータは実行時引数かファイルか標準入力で与える必要があるが，実行時引数は初心者にとってハードルが高い．ファイルで与える場合は，ファイル入出力の単元以降とする必要がある．標準入出力は比較的早いタイミングで単元を済ませることができるため，これをベースとするのがよい．また，実行時の標準出力を指定の名前のファイルへ保存することで，実行と採点を切り離して考えることができる．

5 については，第 2.2.3項および第 2.3節で述べたように，GUI アプリの採点を省力化するためである．MP4 などの純粋な動画形式を利用しない理由は，現状で音声が不要であること，実時間ベースでなくてもよいこと，実装に難があることが挙げられる．要点について自動・短時間で切り替わる形なら，比較・採点が短時間で済ませられる．

6 については，答案に自由度を持たせ

ておきたいためである．詳細については前身のシステム[1)]を参照されたい．

5.2. タイプ 2: CLI アプリ提出時の動作

タイプ 2 の課題，つまり，アプリのファイルを受け取って CLI 上で自動実行および採点する場合の動作は，次のような流れである．

1. ファイルを受け取り，ZIP ファイルの場合は展開
2. ファイルを解析して自動採点に必要な情報を抽出
3. 提出された CLI アプリを入力用データファイルに基づいて自動実行
4. 出力および諸情報に基づいて自動判定しつつ，必要に応じてコメント

タイプ 2 標準スクリプトの前半をコード 1に示す．1〜5 行目では，変数を初期化し実行フォルダ（課題別・ユーザ別フォルダ）へ移動する．12〜17 行目では，解析に先立って，isort, autoflake, black を用いてソースコード群を自動整形し，コメント行を削除しつつファイルを output_pc.txt として結合する．このとき，ファイル名が「myapp.py」のものを予め「_.py」に変更しておくことで，モジュールファイルが同梱されていても課題のメインプログラムを先頭にして結合できる．

19〜24 行目では，まず，output_pc.txt を解析してクラスや関数・メソッドの情報を抽出し，output_pc2.txt として保存する．その具体例をコード 2に示す．

ここで，「class」の後はクラス名を，「def」の後はメソッド名を表し，「class ~; def」の後は関数名を表している．続いて，課題バンクフォルダ内に output_pc2.txt が存在している場合，diff コマンド[注 19)]を用いて比較した結果を diff_pc2.log として保存する．これにより，クラスや関数・メソッドについて指示に従っているかを確認できる．

タイプ 2 標準スクリプトの後半をコード 3に示す．26〜34 行目では，課題バンクフォルダ内に標準入力用データファイル「input*.dat」（*はワイルドカード）が存在する場合に，その個数分の自動実行を繰り返す．その際の標準出力は，装飾用の制御コードを取り除いた上で「output*.dat」（*は input に対応）保存される．対応する標準出力比較用のファイルが存在する場合は，diff コマンドを用いて比較した結果を diff_res.log として保存する．これにより，実行結果が想定のものであるかを確認できる．なお，実行には各 5 秒のタイムアウトを設定し，出力ファイルサイズにも上限を設けている．アプリの無限ループやページに実行結果を表示する際のトラブルを回避するためである．

36〜47 行目では，これまでに得られた

コード 1: 標準の自動採点スクリプト前半（タイプ 2，タイプ 3 共通）

```
# 変数を初期化し実行フォルダへ移動
d_sh=$(pwd)
p_id=$(basename ${d_sh})
flag_k=0
cd ../../${1}/${2}

# ソースコード群を空の一時フォルダへコピー
rm -fr /pytmp/${2} 2>> error.log
mkdir -p /pytmp/${2}
cp -fr *.py /pytmp/${2} 2>> error.log

# ソースコード群を自動整形した後に結合
mv /pytmp/${2}/myapp.py /pytmp/${2}/_.py
isort /pytmp/${2}/*.py 2>> pc_fmt.log 1>> error.log
autoflake -ri --remove-all-unused-imports --ignore-init-module-imports --remove-unused-variables /pytmp/${2}/*.py 2>> pc_fmt.log 1>> error.log
black --line-length 159 /pytmp/${2}/*.py 2>> pc_fmt.log 1>> error.log
cat /pytmp/${2}/*.py | sed -e '/^[␣␣]*#/d' -e 's/[␣␣]*$//' 1> output_pc.txt 2>> error.log

# クラスや関数・メソッドの情報を抽出
python3 ${d_sh}/../_default/make_pc2.py < output_pc.txt 1> output_pc2.txt 2>> error.log
if [ -s ${d_sh}/output_pc2.txt ]; then
  flag_k=1
  diff -diN --strip-trailing-cr output_pc2.txt ${d_sh}/output_pc2.txt >> diff_pc2.log
fi
```

結果から評価値を決定している．何らかの比較が行われた場合，それらの差分が存在すれば（差分ファイルが空でないならば）不合格（F），存在しなければ合格（G）とし，比較が行われていない場合は確認中（K）としている．第 2.4節で述べたように，評価値が F の場合，その理由や助言をログファイルに追記している．学習者が課題ページまたは提出後のページでこれを読むことで，修正と再提出を早期に行うことができる．

この標準の自動採点スクリプトによって，入出力の関係やプログラムの構造の大枠のみ確認したい課題のほぼ全てが，比較用のファイルを用意するだけで自動採点できる．

コード 2: output_pc2.txt の例

```
1 class Animal:
2 class Animal; def aruku()
3 class Animal; def naku()
4 class Dog(Animal):
5 class Dog; def ouchi()
6 class Dog; def sanpo(,)
7 class ~; def main()
```

5.3. タイプ 3: GUI アプリ提出時の動作

タイプ 3 の課題，つまり，アプリのファイルを受け取ってディスプレイ上で自動実行・自動操作する場合の動作は，次のような流れである．

1. ファイルを受け取り，ZIP ファイルの場合は展開
2. ファイルを解析して自動採点に必要な情報を抽出
3. 諸情報に基づいて足切りの自動判をしつつ，必要に応じてコメント
4. 提出された GUI アプリを実行し，操作指示に基づいて自動操作
5. 要所での SS 群からアニメーションを生成

タイプ 3 の標準の自動採点スクリプトの後半をコード 3に示す．26〜30 行目では，GUI の自動操作に必要な要素

1. 実行できる
2. Tkinter がインポートされている
3. ウィンドウタイトルに学籍番号が含まれている
4. GUI を起動・待機させている

が揃っているかを確認している．ウィンドウタイトルに学籍番号（学習者の ID）を含ませる理由は，SS を撮るウィンドウを検索するためである．グループワークの課題では全員の学籍番号を併記させることで，いずれの学習者が提出しても SS を撮影できる．

32 行目以降では，第 2.1節で述べたように，これまでに得られた結果から足切りを行うかを決定している．タイプ 2 の場合と同様に，評価値が F の場合，その理由や助言をログファイルに追記している．

タイプ 3 では自動採点スクリプトを呼び出した後に，GUI 自動操作のスクリプトを呼び出す．その前半をコード 5に示す．6〜8 行目では，軽量な openbox[注 20)] 上で GUI アプリを起動させている．28 秒のタイムアウトを設定しているが，立ち上がりで 2 秒のスリープを挟むこと，この後にも幾度かスリープさせることから，自動操作は実質的に 20 秒程度で終える必要がある．

10〜17 行目では，提出者の学籍番号（学習者の ID）を含むウィンドウを探して，scrot コマンド[注 21)] で SS を撮り，

コード 3: 標準の自動採点スクリプト後半（タイプ 2）

```
26 # input*.dat があれば自動実行
27 for f_i in ${d_sh}/input*.dat; do
28   f_o=$(basename ${f_i/input/output})
29   cat ${f_i} | timeout 5 python3 myapp.py | sed -r 's/\x1b\[[0-9;]*[a-zA-
       Z]//g' | head -n 5000 | head -c 256K 1> ${f_o} 2>> error.log
30   if [ -s ${d_sh}/${f_o} ]; then
31     flag_k=1
32     diff -dBwN --strip-trailing-cr ${f_o} ${d_sh}/${f_o} >> diff_res.log
33   fi
34 done
35
36 # 判定
37 if [ ${flag_k} -eq 0 ]; then
38   echo -n "K"
39 elif [ -s diff_pc2.log ]; then
40   echo -e "\nクラスや関数・メソッドを見直してください" >> diff_pc2.log
41   echo -n "F"
42 elif [ -s diff_res.log ]; then
43   echo -e "\n実行結果が異なります" >> diff_res.log
44   echo -n "F"
45 else
46   echo -n "G"
47 fi
```

ss_win.png として保存している．GUI の自動操作が指示されていない場合は，これが起動時の SS となる．ウィンドウが見つからない場合は起動に失敗していると言えるため，その場合にはその旨をログファイルに追記している．

コード 6に，GUI 自動操作スクリプトの後半を示す．19〜38 行目では，課題バンクフォルダに操作指示用のファイル xdotool.txt が置いてある場合に，その指示に従いながら GUI を操作し，その過程をアニメーションとして保存している．xdotool.txt の具体例をコード 7に示す．

空行以外は xdotool 上の指示であり，これによって GUI を操作する．空行の際は，連番を付けつつ scrot コマンドで SS を撮る．こうすることで，課題制作者が意図したタイミングの GUI の様子が画像的に記録できる．

コード 7に対応する SS を図 5に示す．また，対応付けは次のようになる．

0. 操作前（起動時）
1. メニューをクリックしていくこと

コード 4: 標準の自動採点スクリプト後半（タイプ 3）

```
# GUIの自動操作に必要な要素があるか確認
gui_check0=$(python3 -m py_compile myapp.py 2>&1) >> error.log
gui_check1=$(grep -c "^␣*import␣tkinter" output.pc.txt) 2>> error.log
gui_check2=$(grep -c ".title(.*${2}.*)" output_pc.txt) 2>> error.log
gui_check3=$(grep -c '.mainloop()' output_pc.txt) 2>> error.log

# 判定
if [ -s diff_pc2.log ]; then
  echo -e "\nクラスや関数・メソッドを見直してください" >> diff_pc2.log
  echo -n "F"
elif [ -n "${gui_check0}" ]; then
  echo -e "${gui_check0}\nプログラムが起動できないようです" >> error.log
  echo -n "F"
elif [ ${gui_check1} -lt 1 ]; then
  echo -e "\nTkinterを利用してください" >> error.log
  echo -n "F"
# 以降省略
```

コード 5: GUI の自動操作用スクリプトの前半

```
# 変数を初期化し実行フォルダへ移動
d_sh=$(pwd)
p_id=$(basename ${d_sh})
cd ../../${1}/${2}

# 軽量なopenbox上でGUIアプリケーションを起動
timeout 29 openbox 2>> error.log 1>>e1.log &
timeout 28 python3 myapp.py 2>> error.log 1>>e1.log & sleep 3 2>> error.
 log 1>>e1.log

# タイトルに基づいてウィンドウを探し，スクリーンショット
w_id=$(xdotool search --onlyvisible --name ${2}) 2>> error.log 1>> e1.log
if [ -z "${w_id}" ]; then
  echo -e "\nプログラムまたはGUIが正しく起動していません" >> error.log
fi
xdotool windowactivate ${w_id} 2>> error.log 1>> e1.log
xdotool windowmove ${w_id} 0 0 2>> error.log 1>> e1.log
scrot -ubz ss_win.png 2>> error.log 1>> e1.log
```

コード 6: GUI の自動操作用スクリプトの後半

```
19 # GUIの自動操作が指示されている（xdotool.txtがある）場合
20 if [ -s ${d_sh}/xdotool.txt ]; then
21
22   # 行ごとにxdotool向けの命令を実施するが，空行の場合はスクリーンショット
23   count=1
24   while read line1; do
25     if [ "${line1}" ]; then
26       xdotool ${line1} 2>> error.log 1>> e1.log
27     else
28       scrot -ubz /pytmp/${2}/ss_${count}.png 2>> error.log 1>> e1.log
29       count=$((count+1))
30     fi
31   done < ${d_sh}/xdotool.txt
32
33   # 画像をトリミングしてアニメーションを生成
34   wh_tmp=(‘identify -format ’%w␣%h’ ss_win.png‘)
35   mogrify -extent ${wh_tmp[0]}x${wh_tmp[1]} /pytmp/${2}/ss_*.png
36   cp ss_win.png /pytmp/${2}/ss_0.png 2>> error.log 1>> e1.log
37   apngasm ss_win.png /pytmp/${2}/ss_*.png 1 2 2>> error.log 1>> /dev/null
38 fi
39
40 # ウィンドウを閉じて終了
41 xdotool sleep 1 windowkill ${w_id} 2>> error.log 1>> e1.log
42 sleep 1
43 exit
```

でファイル読み込みダイアログを開き，ファイル名を入力

2. Enter キーで処理を進める
3. テキストエリアをクリックして文字列を入力
4. メニューをクリックしていくことでファイル書き込みダイアログを開き，ファイル名を入力
5. Enter キーで処理を進める

1 コマ目と 4 コマ目ではダイアログが立ち上がってアクティブになっているため，それらが撮られている．また，トリミングにより右側が切れている．

コード 6の 33〜35 行目では，ImageMagick で SS 群をトリミングし，そのサイズを ss_win.png に揃えている．37 行目では，APNG Assembler[注 22)] で SS 群からアニメーションを生成し，ss_win.png を上書きしている．アニメーションは，0.5 秒ごとに画像を送りつつループ再生する形である．こうすることで，GUI の

コード 7: xdotool.txt の例

```
mousemove 20 40
click 1
mousemove 20 70
click 1
type output_pc2.txt

key Return

click 1
type hogehoge
key Return

mousemove 20 40
click 1
mousemove 20 100
click 1
type output.dat

key Return

```

自動操作の様子をブラウザ上で素早く確認できるようになる.

6. システムの有用性の検証

6.1. 検証方法

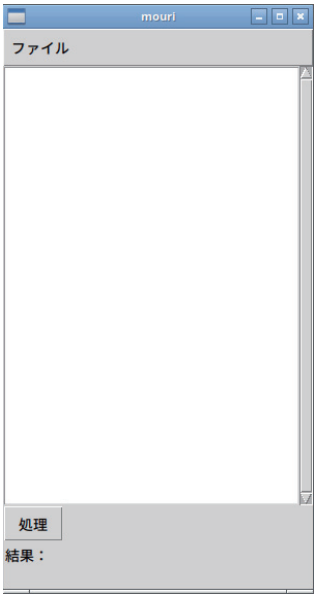
本システムの開発目的は，特に GUI アプリが提出された際の，採点者の負担軽減と学習者へのフィードバックの早期化である．両方に共通する指標として，採点に要した時間から本システムの有用性を測ることとした．具体的には，GUI アプリを提出する課題について，次のような調査を行い，結果を比較した.

1. 支援あり：答案一覧ページの表示から全件の評価の登録までにかかった時間を計測
2. 支援なし：答案一覧ページの表示から，答案のダウンロード・実行・操作を行い，全件の評価の登録までにかかった時間を計測

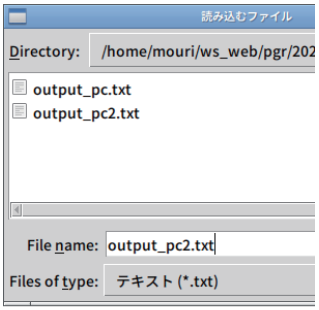
ここで，ブラウザ上のアニメーションにて不合格が疑われる答案に対しては，コメントのため支援ありの場合もソースコードの表示または提出物のダウンロード・実行・操作を行った．なお，実際に授業で提出された答案群を用いた関係上，個人情報保護の観点から筆者が採点者と計測者を兼務した．第 4.3節で述べたように，採点結果を課題ごとにまとめて入力する形をとっている関係上，答案 1 件ごとの時間計測は行っていない.

6.2. 調査結果

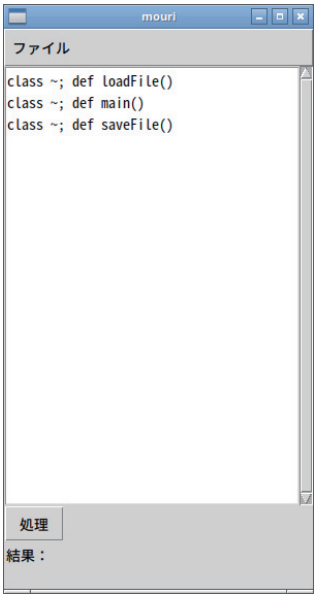
前述の調査を複数の課題について実施した．各課題の採点時間の総和の比較を表 2に示す．支援機能を用いることで，いずれの課題も 30%未満の時間で一通りの採点を終えられている．ここで，被験者・採点者である筆者は支援なしの採点を 8 年間程度経験しており，支援なしの採点を開発期間を含めて実験時点で 1 年



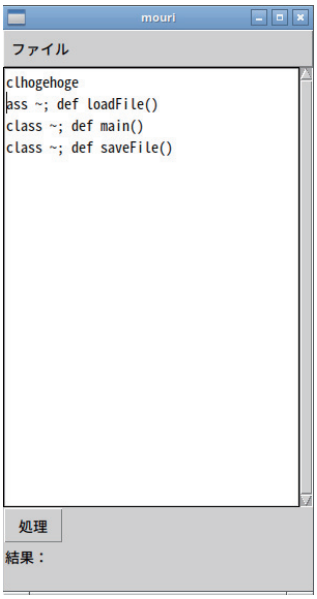
（起動時）



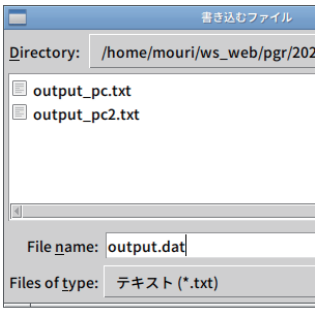
（1 コマ目）



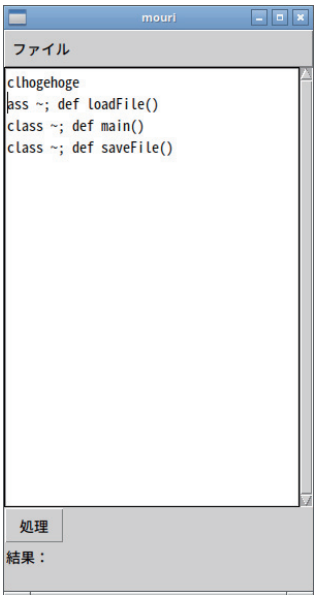
（2 コマ目）



（3 コマ目）



（4 コマ目）



（5 コマ目）

図 5: スクリーンショットの例（コード 7に対応）

表 2: 各課題の採点時間の総和 [秒]

課題	件数	支援あり	支援なし	比
a064	18	85	386	0.220
a071	19	117	436	0.269
a072	18	94	345	0.272
a074	16	103	390	0.264
a075	13	81	477	0.170
a076	10	59	262	0.225

間程度経験している．本実験では，被験者が支援の有無に関わらず慣れているために，操作ミスなどが無く安定した結果が得られたと言うことができる．逆に，採点や本システムの支援に不慣れな利用者の場合には，採点時間の揺らぎが大きくなりうる．特に，不合格が疑われる答案を採点する際には，支援ありの場合もソースコードの表示または提出物のダウンロード・実行・操作を行うため，採点のフローが変化する．その判断と対処に慣れていない場合は，アニメーションの再確認や操作ミスなどにより，支援なしの場合より時間を要する可能性もある．逆に，正解の動作が限定的で，かつ，不合格が疑われる答案が少ない場合，採点時間の比はより 0 に近づくと考えられる．

比が最も小さい課題 a075 は図 5にも示した課題であるが，ダイアログを利用する点とファイル入出力を伴う点で，支援なしの場合には大きな時間を使うことになった．支援ありの場合は，第 4.3節で述べた，出力されたテキストファイルの中身をブラウザ上に表示させる機能を利用して確認できるため，短時間で採点できたと言える．

6.3. 考察

本システムを利用することで，採点時間はもちろん，採点時のストレスが大幅に減ったことを実感している．授業中に採点する場合，その間は説明や学生のケアができないため，時間がかかると「早く終わらせたいという」プレッシャーが生じる．その中で，ファイル入出力を扱う課題 a075 のようにステップ数が多い場合は，心理的負荷が特に大きいものであった．それらが軽減されることの意義は大きいと言える．

学習者側の反応としては，以前に授業の SA[注 23)]から，次のような意見を得ている．

> 先生が質問対応に早く入れるようになって良かった．自分の時には動画ではなく，提出したプログラムがちゃんとしているか不安だった．提出した側でも確認できる点が良い．正解の動画と並んで表示されていると，もっと分かりやすくて良い．

図 3および図 4のように，GUI の場合でも提出者にも採点者にも正解プログラムの

実行結果と並べて表示する仕様にしたのは，この意見を受けてのことである．SA以外からは，システムに対する意見が特に出ていないが，学習者にも実益があったのであれば，本システムの導入に意義があったと言える．

6.4. 課題点

システムそのものではないが，GUIの自動操作について適切な指示を作成する労力が大きな点が，本システムの課題と言える．まず，普段使っているウィンドウマネージャがopenboxではなく，作成に際して切り替える必要がある．また，クリックなどの際の画面上の位置をはじめ，操作について丹念に検討・調査することが必要である．加えて，xdotoolについての知識も必要であり，指示ファイルの作成には試行錯誤も必要である．つまり，他者はもちろん，筆者自身が課題を追加・編集するにも労力が大きくなってしまう．コンテンツを充実させるためには，自動操作の指示の作成支援機能が必要であろう．

前向きな課題としては，自動採点の対象を増やせる可能性が挙げられる．自動化アニメーションやスクリーンショット群について，正解プログラムのものと上手く比較できれば，それに基づいて自動採点ができる可能性がある．現行より負担軽減やフィードバックの早期化を目指す場合，そのような機能が必要であろう．

7. まとめ

本稿では，GUI自動操作とスクリーンショットに基づくGUI採点支援機能を有したPythonプログラミング実習用システムについて，設計・構成を説明するとともに，採点時間を比較することで本システムの有用性を示した．まず，採点者にかかる負担について分類し，ぞれぞれの軽減について述べた．続いて，外部・内部のシステム設計と自動採点・GUI採点支援機能について，抜粋しながら詳細に示した．そして，採点時間が短縮されたことから本システムの有用性を示した．

今後の課題としては，コンテンツを充実させるためにはGUI自動操作の指示の作成支援機能の開発が挙げられる．また，さらなる負担軽減・フィードバック早期化を目指すためには，画像ベースでの自動採点についての検討が挙げられる．

謝辞

本稿の執筆にあたり，本学国際コミュニケーション学部の梅垣敦紀教授に作成して頂いたupLaTeXのクラスファイル[4)]を利用し，またその際に様々なご助力も頂いた．同氏に厚く感謝を申し上げる．

注

注 1）Aizu Online Judge,
https://judge.u-aizu.ac.jp
注 2）Progate, https://prog-8.com
注 3）paiza, https://paiza.jp
注 4）AtCoder, https://atcoder.jp
注 5）Document Object Model のことであり，HTML などのドキュメントの要素を木構造で表現し操作可能にする仕組み.
注 6）Seleinum,
https://www.selenium.dev
注 7）Ranorex,
https://ranorex.techmatrix.jp
注 8）xdotool,
https://github.com/jordansissel/xdotool
注 9）PyAutoGUI,
https://pyautogui.readthedocs.io
注 10）ソースコードや課題バンク等は下記の URL にて公開している.
https://halo.aichi-u.ac.jp/~mouri/works/programming_training_system_python.zip
注 11）javap, Java の SDK に付属する，バイトコードで書かれたクラスファイルを解析・逆アセンブルして人間が理解できる文字列に変換するツール.
注 12）isort,
https://pycqa.github.io/isort/
注 13）autoflake,
https://github.com/PyCQA/autoflake
注 14）black,
https://github.com/psf/black
注 15）Python として実行可能な圧縮ファイル用の拡張子．実際には特殊なファイルの追加後に ZIP 圧縮されていることを表す.
注 16）timeout, プログラムの実行に時間制限を設ける Linux コマンド.
注 17）xvfb-run, 仮想のディスプレイフレームバッファを生成してプログラムを実行するためのコマンド.
注 18）Animated Portable Network Graphics のことであり，画像形式である PNG を動画に拡張したもの.
注 19）diff, 2 つのテキストファイルを比較し，差分を出力する Linux コマンド．空白や改行を無視した比較も可能.
注 20）openbox, Linux 用のウィンドウマネージャ．シンプルかつ軽量で立ち上がりが早い．https://openbox.org
注 21）scrot, スクリーンショットを撮るための Linux コマンド．アクティブウィンドウに対しタイトルを含めた撮影も可能
注 22）APNG Assenbler, PNG ファイル群を 1 つの APNG ファイルに変換する.
https://apngasm.sourceforge.net
注 23）SA, 有償で授業の手伝いをする学生.

参考文献

1） 毛利 元昭: “JAR ファイルを対象とする自動採点機能を有した Java プログラミング実習用システムの構築と運用”, 愛知大学情報メディアセンター紀要「COM」, Vol. 31, No. 1, pp.11-35, 2022.

2） 立石 良生, 井上 潮: “模範解答を利用した GUI プログラミング課題の自動採点システム”, DEIM2020 最終論文集, I1-1, 2020.

3） 水谷 晃三: “Vision Language Model を用いたプログラミング課題の自動採点システムの設計と予備的評価”, 情報処理学会 情報教育シンポジウム SSS2025, pp.72-79, 2025.

4） 梅垣 敦紀: “愛知大学情報メディアセンター紀要「COM」への upLaTeX での投稿”, 愛知大学情報メディアセンター紀要「COM」, Vol. 34, No. 1, pp.1-10, 2025.

Title: **Development of Python Programming Training System with GUI Scoring Support Function Based on GUI Automatic Operation and Screenshots**

Author: **Motoaki MOURI**