クラウドを利用した対話的なプログラミング教育環境とその評価手法の提案

○桑田 喜隆[†1] 石坂 徹[†1]政谷好伸[†2] 長久勝[†2] 横山重俊[†2†3] 浜元信州[†3]
[†1] 室蘭工業大学
[†2] 国立情報学研究所
[†3] 群馬大学

Proposal of Interactives Education Environments for Programming on Cloud Services and the Evaluation Method

Yoshitaka Kuwata[†1], Toru Ishizaka[†1], Yoshinobu Masatani[†2] , Masaru Nagaku[†2], Shigetoshi Yokoyama[†2†3], and Nobukuni Hamamoto[†3]
[†1] Muroran Institute of Technology, Japan
[†2] National Institute for Informatics, Japan
[†3] Gumma University, Japan

概要

近年，Jupyter Notebookを初めとした対話的な計算機環境の活用が注目されている．対話的に試行錯誤を行い，その経過や結果を記録として残すことができるため，研究成果の共有目的のみならず，教育目的でも効果が期待される．
本稿では，対話的なプログラミング教育環境の有効性を示すため，クラウドを利用して環境を構築し，その環境下で有効性を評価する方法を提案する．

Abstract

Interactive environments, such as Jupyter Notebook, becomes popular in scientific research and in large-scale data analytics. These environments are effective also for learning programming, because students can try and error on them, and the results are stores as file for future references.
In this paper, we propose to make use of Jupyter Notebook to learning programing. We also discuss the evaluation method of them.

1. はじめに

1.1 プログラミング教育の必要性

近年，ICT を活用するための基礎教育が注目されている．

文部科学省で作成した新学習指導要領（情報教育・ICT 活用教育関係）[1)]では，小・中・高等学校共通のポイントとして情報活用能力を，言語能力と同様に「学習の基盤となる資質・能力」と位置付け，学校の ICT 環境整備と ICT を活用した学習活動の充実を明記している．さらに，これからの時代に求められる資質･能力として「プログラミング的思考」が必要であるとしている．今後，新学習指導要領が普及するに従い，計算機を利用するための論理的な思考を身につけた人材が育成されることが想定される．

他方，高等教育のプログラミング教育については，明示的に議論の対象とされてはいないものの，初等・中等教育で習得した基礎の理解を完全なものとし，更に専門教育の必要性に応じた内容を扱う必要があるものと考えられる．たとえば，様々な計算アルゴリズムや，それらの計算のためのデータ構造，計算量の考え方などは，計算機を利用する上で必須の知識であると考えられる．

本稿では，高等教育の共通科目として，プログラミングを行う場合を取り上げ，その実施方法について議論する．特に，大学の共通科目として理系学部学生のプログラミング教育を想定して，効果的な実施方法と評価方法について論じる．理系

[1] Yoshitaka Kuwata
室蘭工業大学
北海道室蘭市水元町２７−１
kuwata@mmm.muroran-it.ac.jp

でもプログラミングに関する扱いが分野によって異なり，プログラミング言語，言語処理系，プログラミング環境，利用ライブラリ等が異なるため，共通の課題としてプログラミング教育を論じることが難しい．

本稿では，Jupyter Notebook を利用した対話的な環境を利用したプログラミング教育の方法を提案する．また，対話的環境の利点を生かし，学生の理解度を把握する目的で，学生の試行錯誤の履歴を把握する方法を提案する．

1.2 プログラミング教育に関する仮説

IT は進歩が早いため，プログラミングを修得しても，すぐに役に立たなくなるのではないかという懸念がある．

表１に年代ごとに教育に使われたプログラミング言語，プログラミング環境および計算機環境の例を示す．

表１　教育用プログラミング言語，環境の例*

年代	プログラミング言語	プログラミング環境	計算機環境
1960-70	FORTRAN, LISP, C	机上コーディング,	汎用機，ミニコン
1980	C, BASIC, Pascal	エディタ＋コンパイラ	WS, 8 ビット PC
1990	C/C++, Java, Perl	グラフィカルプログラミング	WS, PC
2000	C#, Java, PHP	IDE（統合環境）	同上
2010	JavaScript, Swift, Python	インタープリタ	クラウド環境

プログラミング言語，環境は計算機環境の発展に合わせて大きく変化していることが分かる．これに対して，筆者らは，以下の仮説を立てたる．

【仮説１】プログラミング言語やプログラミング環境は時代によって変化する．従って，これらの知識は時間が経つと陳腐化する可能性がある．

【仮説２】アルゴリズムやデータ構造などのプログラミングの基礎は普遍的な要素を含んでおり，プログラミング言語や環境と異なり，時間を経ても役に立つ．

類似の話題として，大岩 [4]は次のように述べている．

> *（前略）驚くべきことに日本では今に至るまで「プログラミング」教育がプログラミング言語教育にとどまっている．それどころか，その方法が，サンプルプログラムをそのまま入力して実行し，その結果を確認するという「写経型学習過程」による教育が一般的になっているようだ．*

ここで議論されている「写経型学習過程」は，プログラミング演習で見本のプログラムを打ち込み，それを実際に動かすことを繰り返すことで，演繹的に一般化したプログラミングの知識を習得する方法である．プログラミング言語の文法などを覚えることに適するが，プログラミングの原理や考え方に関しては，別に教授することが必須であると考える．

また，岡本 [5]は，写経型学習過程における共通の「つまずき」を類型化して分析している．

2. 対話的なプログラミング環境の提案

2.1 前提条件

本稿では理系学部学生のプログラミング教育（大学一年生全員）を対象として議論する．前述の教育課程の変更に伴い，今後状況が変化することが期待されるが，現在のところ，受講者にはプログラミング未経験者も多く含まれる．このため，学生の理解度にばらつきが大きい点が特徴となっている．

プログラミング教育の達成目標を以下に設定する．

(1) アルゴリズムやデータ構造などプログラミングの基礎を理解する．
(2) 基礎的なプログラムを書くことができるようになる．
(3) 上記を達成するために，プログラミング言語やプログラミング環境について必要なことを理解する．

2.2 対話的プログラミングの実施手法

本稿で提案する対話的プログラミングは授業の中で演習を繰り返し，試行錯誤によって概念を修得することを仮定している．しかし，写経型学習過程とは異なり，概念を説明した上で理解を進めることを目的とした演習を実施する．同時に演習

* プログラミング言語は(文献 3)を元に作成．教育で使われた時期はプログラミング言語が発表された時期とは異なることに注意されたい．

結果を評価し，理解を助けるためのフィードバックを行う．

対話的プログラミング環境を使った授業の進め方の例を図１に示す．

- 教員は，学生が自ら試行錯誤できるプログラミング環境を用意する．（対話的プログラミング環境）
- 教員は事前学習教材を準備する．
- 学生は事前学習を行い，予め概念を自分なりに理解しておく．
- 教員は授業の中で基本的で重要な概念を教授する．
- 学生は対話的プログラミング環境を使って演習課題を解く．不明な点があれば，教員や TA，他の学生に質問し理解を深める．
- 学生は授業の最後に確認問題を解いて，提出する．
- 教員は確認問題を評価し，次回以降の授業時に学生にフィードバックする．

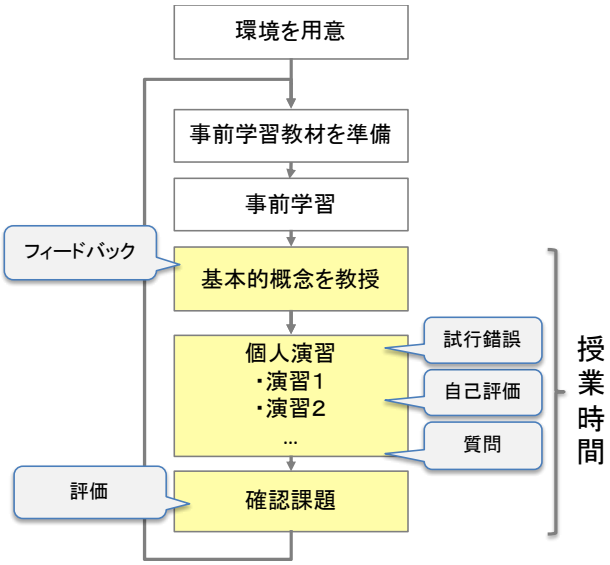

図1　授業の進め方の例

2.3 対話的プログラミング環境の比較

対話的プログラミング環境を利用することで，コードの編集方法やコンパイル方法などのプログラミング環境に関する操作や前提知識の習得が少なくなるため，演習に対する敷居が下がることが期待される．

表２に対話的なプログラミング環境と非対話的なそれとの比較を示す．

表２　プログラミング環境の比較

	対話的な環境	非対話的な環境
例	Python (Jupyter Notebook)	C 言語
実行サイクル	入力→実行	編集→コンパイル→実行
メリット	・プログラミング環境の習得が容易 ・実行サイクルが短い	・実行前にプログラム全体の静的な確認ができる ・実行速度が速い
デメリット	・静的にプログラム全体の確認はできない ・実行順序や内部状態によって実行結果が異なる	・プログラミング環境の習得に時間がかかる ・実行サイクルが長い

3. 課題

前章で提案した対話的なプログラミング教育環境の有効性の検証を行う必要がある．検証を行うために，以下の実現が必要になる．

(i) 教員により授業ごとにカスタマイズ可能なプログラミング環境
(ii) 教員によるプログラミング環境の一元管理および，進捗状況の把握
(iii) 効果測定のための指標の設定とその取得
(iv) 対話的プログラミング教材

4. Jupyter Notebook を使ったプログラミング教育

4.1 Jupyter Notebook とは

Jupyter Notebook[2]は Web ベースの対話的な計算機環境である．利用者はブラウザからプログラミング言語を通じて，計算を行うことができる．近年，科学技術計算やデータ解析，データの可視化といった分野で利用が広がっている．

Python, R, Julia, Scala といった複数の言語処理系がサポートされているが，自分でプログラムを作らない場合でも，大規模計算や機械学習などのライブラリを利用して計算を行うことが可能である．計算式だけでなく，その説明をマークアップ言語で記述したり，グラフや画像イメージ，ビデオなどを Web ページ上にインラインで挿入することで，計算結果を直感的に理解しやすい形式で記録することが可能である．

また，Jupyter Notebook で作成したコンテンツ

(Notebook)は，さまざまな形式で公開し，第三者と共有することも容易である．

4.2 Jupyter Notebook を使ったプログラミング教材の例

Jupyter Notebook は対話的にプログラムを入力し，その結果が可視化されることから，科学技術計算だけではなく，プログラミング教育にも向くと考えられる．

図２に本稿で述べたプログラミング教育用に向けて準備中の Jupyter Notebook を利用したプログラミグ教材の例を示す．

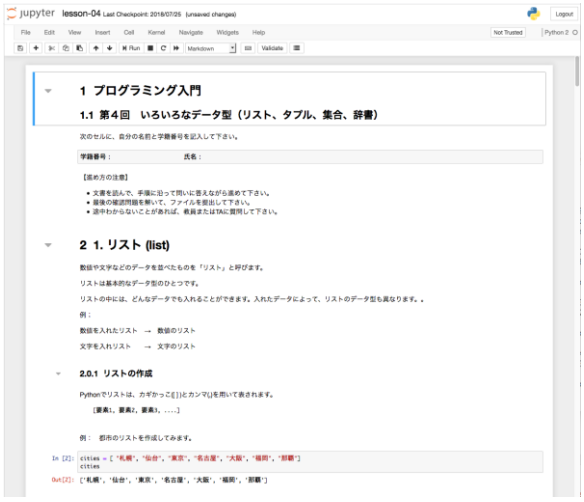

図２ Jupyter Notebook を使った教材の例

Jupyter Notebook では実行可能なセル（コードセル），および説明用の文章のセル（マークダウンセル）を利用してテキストを記述する．

プログラミング教材では，次のように配置する．

(1) 説明
演習内容やその前提知識の説明を，マークダウンセルに記述する．また，学生の理解補助のため，数式や図を併用する．

(2) 記述済みのコードセル
例題として，記述済みのコードを含むセルを用意する．学生はコードをその場で実行して，結果を確かめることができる．また，学生が内容を変更して再実行することで異なる結果が得られることを確認できる．
更に，故意にエラーの出るコードを用意しておき，学生に原因を考えさせる応用も考えられる．

(3) 部分記述済みのコードセル
コードの一部のみを記述しておき，未記入の部分を学生が記述することでプログラムを完成する．
プログラムが複雑になる場合には，複数のステップ（セル）に分割してコードを作成する手法も有効であると考えられる．

(4) 未記入のコードセル
自由にコードを書けるように，未記入のセルを用意する．これまでの説明を理解したかどうかを確認するために利用すると効果的である．例えば，単元ごとに設ける提出課題は未記入のコードセルを利用する．

4.3 対話的プログラミングのプラットフォーム

授業のニーズに応じて教員がプログラミング環境をカスタマイズし，その効果を把握することのできるプラットフォームが望まれる．そこで，クラウドコンピューティング技術を利用したプラットフォームを提案する．

図３に対話的プログラミング教育のためのプラットフォームの概要を示す．

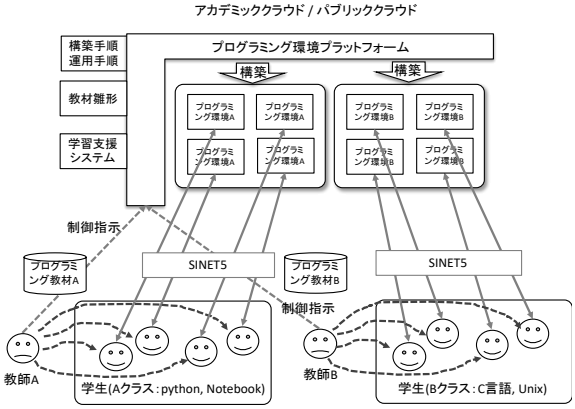

図３ 対話的プログラミング教育のためのプラットフォーム

本提案では，学生の利用するプログラミング環境をクラウド上に用意する仕組みを構築する．コース開設時に教員がコースに合わせて雛形を作成し，雛形をもとに構築したプログラミング環境を学生が利用する．本プラットフォームでは，学生のプログラミング環境を教師が一元管理しているため，学生の状態（課題の進行状況や誤りなど）をリアルタイムに知ることができる．また，授業の進行に応じて教材や課題を学生のプログラミング環境に配布する．学生の活動履歴を一元

的に収集し，授業後に分析することも容易である．

学生用のプログラミング環境の実現には仮想化技術の利用を想定しているが，Docker などコンテナを利用して構築しても良い．

この仕組みは，従来の C 言語の授業など，Jupyter Notebook 以外のプログラミング環境としても利用可能である．異なる授業の比較を行うことで，言語処理系やプログラミング環境の違いによる教育効果の定量的な比較も可能になる．

5. 活動履歴の分析手法

本章では，Jupyter Notebook で作成された教材を使って演習を実施した場合の活動履歴の分析について論じる．

5.1 活動履歴の分析方法

活動履歴には，学生が演習で行った行動が時系列に記録される．Jupyter Notebook の場合には，以下のような内容である．

(A) Notebook を開く
　ファイル名，使用する言語処理系
(B) Notebook を保存する
　ファイル名，ファイル内容，サイズなど
(C) Notebook を閉じる
　ファイル名，ファイル内容，サイズなど
(D) セルを評価する
　評価したセルの内容及びその評価結果

上記を利用することで，以下の分析が可能である．

(1) 進捗状況の把握：(A)(B)(C)を利用
- ファイル単位の進捗速度
- セル単位の進捗速度

(2) 入力内容の評価：(D)を利用
- 学生の入力した式や関数と模範解答との比較
- 言語処理系から返されるエラーの有無
- 言語処理系から返される出力結果と模範解答の出力結果との比較
- 関数定義などの場合，テストケース評価による正誤判断

ただし，Notebook はセル評価の順番に制限がないため，Notebook に複数含まれるセルのどれを評価したかを正確に把握する方法がない．例えば，セルを上から順番に評価しなかった場合や，複数回同じセルを評価した場合には，単純に活動履歴とセルの対応を取ることは出来ない．Jupyter の Jupyter の拡張機能である，JLC wrapper[6]を利用した場合，履歴情報から取得可能である．

5.2 分析結果の活用方法

目的に応じて，分析のタイミングおよび分析単位，結果の活用方法を選択する必要がある．

(1) 分析のタイミング
　リアルタイム分析はその場で分析を行う方法である．分析結果をその場で学生にフィードバックする場合などに利用される．
　事後分析は集積した情報をもとに分析を実施する方法である．次回の授業時の教師からの補足説明やフィードバックに利用される．

(2) 分析の単位
　学生ごとに分析する方法と，クラス全体を分析する方法がある．

分析タイミング及び分析単位の組み合わせによって，以下の活用方法が考えられる．

① 学生ごとのリアルタイム分析
　入力の正誤を示して再考を促す．
　問題を解く過程を評価する．
　採点する．
　入力の誤りに対してヒントを与える．
② クラス全体のリアルタイム分析
　クラス全体の進捗状況を示す．
　クラスの中の位置付けを示す．
③ 個人ごとの事後分析
　個人ごとの理解度を示す．
　教材テーマごとの理解度を示す．
④ クラス全体の事後分析
　教材テーマごとのクラス全体の理解度を分析する．
　例：よくある間違え箇所を分析する．

また，分析結果の開示方法について，以下の３種類，およびその組み合わせが考えられる．

(イ) 教員に開示する
(ロ) 該当する学生に開示する
(ハ) クラス全体に開示する

例えば、クラスの中での進捗度合いを，(a)教師だけが見る (b) 本人と教師だけが見る (c) 全員が見る，という選択が可能であるが，その意図および効果が異なる．

5.3 既存のプラグインの利用

前節であげた機能の一部は，Jupyter プロジェクトの作成された nbgrader[7]によって提供されている．

nbgrader の機能について，以下に概要を示す．

(a) 穴埋め式の課題生成
テンプレートを使った穴埋め式の課題の生成が可能である．

(b) 正解・不正解の判断
テストケース（プログラムへの入力と出力結果の組）を記述しておくことで，コードを自動実行して正解かどうか確認することができる．

(c) 自動/手動採点
テスト結果に基づき課題に自動的に採点する機能．手動で採点することもできる．

(d) 学生によるコメントや質問の記述
教員向けに，コメントを自由に記述することができる．

6. 考察

6.1 実装方法に関する留意事項

前章で活動履歴の分析手法に関して述べた．提案した分析手法は，既存のプラグインモジュールでは実装されていないため，何らかの形で実装することが必要である．

Jupyter Notebook は開発途中であり，頻繁にリリースが行われているため，プログラムを作り込んでも陳腐化が早い．このため，なるべく既存のプログラムを活用し，提案手法の有効性検証を優先して実施することが必要であると考える．

また，学習支援システムである Moodle を使い授業を行うため，Moodle[8),9)]とも連携することが必要がある．LTI(Learning Tools Interoperability)を Jupyter に実装し Moodle と連携する方法や，学認 mAP を使いシングルサインオンできるようにする方法が考えられる．

システム運用に Jupyter Notebook を利用する提案もあり [10)]，その成果を活用する予定である．

6.2 履歴情報収集および分析に関する配慮

本稿では学生の状態を分析し，フィードバックを行う方法を提案している．このためには，予め情報を収集し分析することを学生に説明し理解してもらった上で実施することが必要となる．

また，情報セキュリティ上の配慮も必要である．クラウドコンピューティングサービスで外部業者を利用する場合，外部へのデータ流失などのリスクについてもアセスメントと対応計画の立案が必須である．また，履歴データを他の目的に利用できないように，外部へ持ち出しすことができないような仕組みを構築することが望ましい．

7. まとめと今後の課題

本稿では，理系学部学生全員が受講することを前提に，対話的なプログラミング教育環境を提案した．また，その有効性を評価するための方法について提案した．

本稿で述べた方法で，Jupyter Notebook を使った環境を構築し，プログラミング教材を準備中である．実際の授業を行ってその効果を測定することが課題である．また，授業実施に先立ち，模擬的な評価等によってその効果の確認をすることを検討中である．

本研究は JSPS 科研費（JP18K11561)の「クラウドを活用したプログラミング演習環境に関する研究」の助成を受けたものである．

A. 参考文献

1. 文部科学省, 小学校プログラミング教育の手引き（第一版）, http://www.mext.go.jp/a_menu/shotou/zyouhou/detail/1403162.htm (2018/8/31 参照)
2. Project Jupyter, Project Jupyter Homepage, http://jupyter.org/ (2018/8/31 参照)
3. Wikipedia, プログラミング言語年表, https://ja.wikipedia.org/wiki/プログラミング言語年表 (2018/9/9 参照)
4. 大岩元,プログラミング教育の社会的意義　ー設計から始めるプログラミングー,情報教育シンポジウム 2016 論文集，Pp. 122 - 128，2016-08-15, http://id.nii.ac．jp/1001/00174191/
5. 岡本雅子,はじめてのプログラミングとつまずき, 情報処理 56 巻 6 号,Pp. 580 - 583, 2015-05-15, http://id.nii.ac.jp/1001/00141758/
6. 国立情報学研究所, Jupyter-LC_wrapper, https://github.com/NII-cloud-operation/Jupyter-LC_wrapper （2018/9/14 参照)
7. Project Jupyter, nbgrader, https://nbgrader.readthedocs.io/en/stable/
8. Moodle プロジェクト，https://moodle.org/, (2018/9/14 参照)
9. 桑田喜隆, 石坂徹, 合田憲人, 竹房あつ子, 横山重俊, 浜元信州，パブリッククラウドを使った Moodle の運用評価, 317-P, Moodlemoot 2018
10. Literate Computing for Reproducible Infrastructure,https://literate-computing.github.io/ (2018/9/14 参照)

※ 記載されている会社名，商品名，又はサービス名は，各社の商標又は登録商標です．