初学者向けProcessingプログラミング環境における コード補完機能の導入と実践

三浦 元喜[1,a)]

受付日 2018年6月4日, 採録日 2018年7月3日

概要：ブラウザのみで利用できる Web IDE は，学習者の環境構築にかかる手間を軽減でき，気軽に利用できるため，初学者に適している．しかし，通常の Web IDE では，プログラムをテキストで記述するため，学習者のタイピングスキルが成績に影響する可能性がある．そこで，我々はタイピングスキルの影響を軽減しつつ，学習者の利便性を向上させるため，自動補完機能による入力支援を Web IDE に導入した．講義や試験における試用の結果，学習者の利用頻度の高い補完候補は if 文や for 文，画面描画関連の関数呼び出しであることを確認した．if 文や for 文のスニペットには，括弧やカーリーブラケットを含んでいたため，補完機能を利用した編集行為の前後は，利用しなかった編集行為の前後と比較して，括弧やカーリーブラケットの入力数が有意に低くなることが明らかとなった．またアンケートの結果，補完機能の使用頻度と，学習者が感じる必要性には，強い正の相関が認められた．

キーワード：プログラミング教育環境，自動補完，タイピングスキル

Introducing Autocomplete Functions to Processing Web IDE for Novice Programmers

Motoki Miura[1,a)]

Received: June 4, 2018, Accepted: July 3, 2018

Abstract: Web-based IDE (WebIDE) is beneficial for the novice programmers because it only requires a web browser to work. However, most of WebIDE requires the novice programmers to write text-based program. Thus the typing skills of the novice programmers affect the performance of the course. To reduce the influence of typing skills, and improve the usability of the novice programmers, we introduced autocomplete functions for our WebIDE. According to the result of experiments, we confirmed that the most frequently-used snippets were if/for statements and several drawing function calls. Since the if/for statements included parentheses and curly braces, our autocomplete functions surely reduces the typing rate of these characters. We also confirmed that the frequency of autocomplete function was highly correlated with the necessity of the function.

Keywords: Programming Environment, autocomplete, typing skills

1. はじめに

ブラウザのみで利用できる Web IDE は，利用者の環境構築にかかる手間を軽減でき，気軽に利用できるため，初学者に適している．そのため，Web IDE を利用したプログラミング環境が多数開発されている [1], [2], [3], [4], [5]．我々も，C 言語に類似した文法を備えつつ，グラフィックスやアニメーションを簡潔に記述することができる Processing 言語に着目し，その Javascript 版の実装である Processing.js[*1] を利用して，大学生を対象とした講義を行ってきた [6].

これらの環境では，おもにテキスト記述によるプログラミングが主流となっている．しかし，テキスト記述によるプログラミング学習を進めるうえでは，一定のタイピング

1 九州工業大学 基礎科学研究系
1-1 Sensui, Tobata, Kitakyushu, Fukuoka, 804–8550, Japan
a) miuramo@mns.kyutech.ac.jp

*1 http://processingjs.org/

スキルが要求されるため，タイピングスキルのレベルによって，プログラミング学習の成果に影響を及ぼす可能性がある．**表 1** は，我々が過去 4 年間で担当した初学者向けプログラミング講義における，タイピングスキルと最終成績の相関係数 (ピアソンの積率相関) を調査した結果である．ここでのタイピングスキルは，2 回目の講義の最初に，学習者が自己申告したタイピングソフト yatt[7] のスコアを 1〜7 の数値で表したものである．1〜7 の数値に対応するスコアの意味と，人数の分布は付録 A.2 の**表 A·1** に示している．開講年度やクラスによって差があるものの，タイピングのスキルと，プログラミング講義の最終成績との相関が高い年もある．しかし，本来タイピングスキルのレベルや優劣が，プログラミング学習成果に影響を与えてしまう状況は望ましいことではない．

表 1 タイピングスキルと成績の相関 (*と**はそれぞれ有意水準 5%, 1%で有意)

年度 (クラス)	受講者数	r	t 値	p 値
2017	45	0.024	t(43) = 0.158	0.875
2016	69	**0.389****	t(67) = 3.457	0.001
2015	72	**0.395****	t(70) = 3.599	0.001
2014 (1)	75	**0.252***	t(73) = 2.223	0.029
2014 (2)	49	0.136	t(47) = 0.943	0.351
2014 (3)	52	0.214	t(50) = 1.552	0.127

タイピングスキルの影響を軽減するためのひとつの方策として，Scratch[8] や Google Blockly[9], [10] といったブロック記述方式を利用することが考えられる．ブロック記述方式を利用することで，学習者はテキスト記述方式におけるミスやエラーを避けることが可能となる．また，学習者はブロック一覧を見ることによって，利用可能なブロックを短時間で確認できる．しかし，ブロック記述方式はエラーを軽減できる反面，テキスト記述方式に比べてプログラミング記述の自由度が低い．また，操作に習熟するには一定の時間と訓練を要する．我々のグループでも，Google Blockly を利用したブロック型プログラミング学習支援を試行した [11] が，学習者がブロック型インタフェースの操作を理解し，問題解決に利用するには 30 分程度の短時間では不十分であった．

そこで我々は，テキスト記述方式のプログラミング環境における，入力エラーやタイプミスの影響を軽減するため，自動補完機能による入力支援を Web IDE に適用した．また，実際の講義を通じ，自動補完機能が初学者にどのように利用されるかを調査した．

2. 自動補完機能とその実装

自動補完機能とは，キー入力の途中で，補完候補を一覧表示したり，それを選択することによって，キー入力操作を省力化する機能である．vi や Emacs といった古典的なエディタをはじめ，Eclipse や Visual Studio といった統合開発環境でも標準的な機能となっており，関数名や変数名の一部を入力し候補から選択することができたり，対応する括弧を自動挿入できたりする．そのため，タイプミスの軽減や編集効率の向上に加え，関数名や変数名に比較的長い名前をつける際の障壁を下げる効果がある．

今回我々が Web IDE に組み込んだ自動補完機能の実装では，学習者が Tab キー[*2]を押下したときにのみ呼び出されるようにした．通常の文字入力を行っているあいだに候補を自動的に表示する方法もあるが，学習者が意図しなくても候補が表示されたり，コードの一部が隠れてしまったりすることが懸念される．そのため，今回は明示的な操作を必要とする方法を採用した．

具体的な動作の例として，[m] キーに続けて [Tab] キーを押下したときの Web エディタ画面を**図 1** に示す．この状態で，学習者はキーボードのカーソルキー上下で候補を選択 (**図 2**(左)) するか，または続けてタイプすることで候補を絞り込む (図 2(中央)) ことができる．候補を選択している状態で，Tab キーまたは Enter キーを押下すると確定となり，図 2(右) に示すように補完入力が完了する．補完入力完了後は，入力の直後にカーソルが移動する．[Tab] キーを押さなければ，補完機能は起動せず，従来のエディタと同じ振る舞いとなる．また，[Tab] キーを行頭や，半角スペースのあとに入力した場合は，すべての候補が表示される．

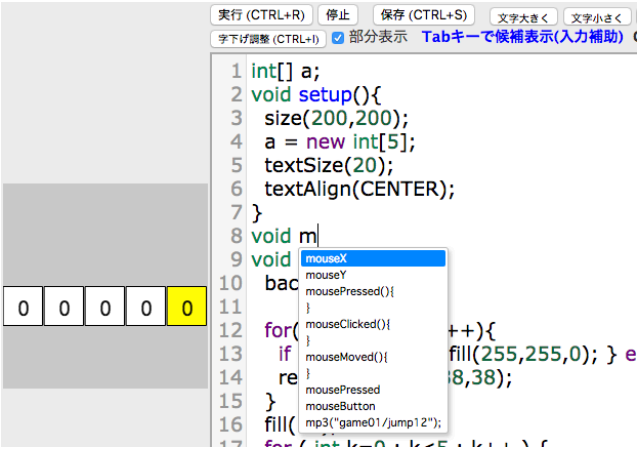

図 1 自動補完機能の呼び出し ([m] キーに続けて [Tab] キーを入力)

自動補完機能の実装にあたっては，CodeMirror エディタ[*3]の show-hint アドオン (ヘルパー) を利用し，Processing プログラミングで利用頻度の高い関数名やキーワード，if 文や for 文のスニペットを補完候補として組み込んだ．ただし，今回の実装では学習者がエディタ内で定義した変数や関数の名前を動的に補完候補に組み込むことはしていない．また，

*2 多くの Linux でデフォルトのログインシェルとなっている Bash の補完操作と一致させた．
*3 http://codemirror.net/

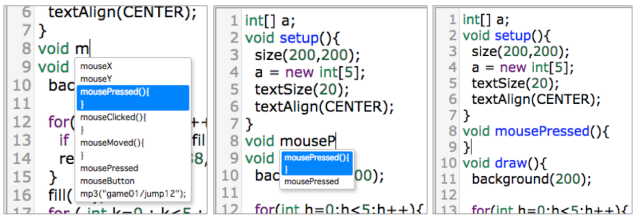

図 2　(左) カーソルキーで選択 (中央) 続くキーで絞り込み (右)Enter キーで確定

補完候補を挿入する箇所のソースコードの文脈については考慮をしていない．付録 A.1 に，今回の実装で組み込んだ Processing キーワード・スニペットの一覧を示す．自動補完機能を有効にした Web エディタは，以下の URL で利用できる．
`https://ist.mns.kyutech.ac.jp/miura/pjs/2018/`

2.1　メリット

自動補完機能の利点について，ブロックインタフェースとの類似点や差異という観点から論じる．

- ブロックインタフェースにおける「機能の一覧表示」に類似した機能を提供できる．
- 命令や関数の利用例を同時に示すことができ，リファレンスの代用となる．
- テキスト形式のプログラミング環境においては，キーボード入力のみで完結できるため，親和性が高い．
- キーワードのタイプミスを防ぐ効果が期待できる．
- 括弧やカーリーブラケット，セミコロンなど，初学者が入力に慣れていない文字を補完するので，タイピングに対する障壁を下げる．

ブロックインタフェースでは，部品同士の接続の可否によって，適合しない記述をフィードバックできるが，自動補完機能ではテキスト記述の段階で，適合性をフィードバックすることは難しい．

3.　運用と分析

実際の学習者が自動補完機能をどのように利用するのかや，その量と効果を検証するため，我々は自動補完機能を組み入れた Web 演習システムを構築し，**表 2** に示すスケジュールで，実際の講義で利用した．学習者の人数は，69 名であった．このうち，操作ログの提供に同意した 48 名のデータのみを分析対象とした．ちなみに，第 5 回の講義以降，自動補完機能は 69 名全員利用可能であった．

操作ログについては，Web IDE の編集操作イベントが 100 個蓄積されたときと，学習者が保存操作を行ったときに，JSON 形式でサーバに送信した．編集操作イベントには，通常のキー押下のほか，カーソルキーによる移動やマウスによるカーソル位置設定，貼り付け操作によるコードの変更と，それらの操作時刻が含まれており，学習者のソースコード編集過程を再現することができる．

表 2　自動補完の導入期間および授業内容

講義回	講義日	設定	講義内容
1	4 月 12 日		導入 (お絵かき)
2	4 月 19 日		関数 (マウスイベント)
3	4 月 26 日	操作ログ記録開始	if 文
4	5 月 10 日		for 文
5	5 月 17 日	自動補完提供開始	アニメーション
6	5 月 24 日		配列と for 文
7	5 月 31 日		中間試験

3.1　操作ログの結果と分析

図 3 に，48 名の学習者のうち，1 回以上補完機能を使用していた 32 名の学習者について，どの課題（練習・演習・試験）で何回補完機能を利用していたかを示す．「練」は練習問題，「演」は演習，「試」は試験を示しており，課題の開示時期によって左から右に並べている．学習者については，利用回数の多い順番に，上から並べている．なお「練 6」は，第 5 回の講義内で，最初に自動補完機能を説明したときの課題である．試験問題の詳細は付録 A.3 の**図 A·1** に示している．これをみると，練習や演習で使用していても，試験のときには使用していない学習者もみられる．補完機能は Tab キーを押下しないと呼び出されないことから，試験のように時間制限がある状況では回答に集中することで補完機能の存在を忘れ，利用しなかった可能性が示唆される．今回の実験では，自動補完を導入した時期が講義の第 5 回目と遅かったため，利用が定着していなかった可能性もある．付録 A.4 の**表 A·2** に，補完機能で選択されたスニペットの利用回数を示す．条件文や，課題・演習に関連する描画命令，繰り返しといった構文が比較的多く利用されていることが確認できた．

学習者	練1	練2	練3	演1	練4	練5	練6	練7	演2	練8	演3	練9	演4	試1	試2	試3	合計
1				11			12		36		11			5	14	21	110
2				5			3	3	14	1	5		29	5	9	4	78
3	12						3			1	3		29	3	7	10	68
4							6		4	2	7			11	7	8	45
5							5	1	4	1	7			7	5	10	40
6							9				4			5	8	12	38
7	4			6			2						6	16	1	1	36
8		4	2	6	2	4	1	2	10		1						32
9							3						1	10	8	8	30
10														3	7	10	20
11														5	13		18
12				3			2	2			2			1	3	1	14
13				3			6					3					12
14				1						3					1	5	10
15														1	1	3	5
16							3							1			4
17							3										3
18							3										3
19							1						2				3
20							1	1	1								3
21													2		1		3
22				2													2
23							2										2
24														1		1	2
25							2										2
26							2										2
27							1										1
28													1				1
29										1							1
30																1	1
31									1								1
32							1										1

図 3　練習・演習・試験ごとの補完機能呼び出し回数

つぎに，補完機能を利用している場合と，していない場

表 3 操作ログにおける，補完機能の使用の有無によるキーの操作量比較 (太字は有意差あり，または有意傾向．**は 1%有意)

操作の種類	補完なし平均値 (標準偏差) n=48	補完あり平均値 (標準偏差) n=36	t 値	p 値
(1) カーソルキー	5.58 (3.73)	5.51 (4.79)	t(60.74)=0.148	0.883
(2) 括弧 (){}	**0.88** (0.37)**	0.44 (0.42)	t(63.70)=3.035	0.003
(3) セミコロン	**0.38 (0.13)**	0.18 (0.23)	t(54.65)=1.970	0.053
(4) BS/Delete	6.63 (2.72)	7.16 (4.43)	t(55.44)=-1.203	0.234
(5) Escape	0.01 (0.04)	0.01 (0.04)	t(67.53)=-0.086	0.932
(6) Enter	0.71 (0.26)	**1.34** (0.78)**	t(44.59)=-3.672	0.001

合とで，キーの操作量に特徴があるかを検証した．48 名の学習者から送信された操作ログについて，送信単位（100 回の操作イベントまたは保存操作）のなかで補完機能を使用している場合と，使用していない場合に分けたうえで，(1) カーソルキー (2) 括弧とカーリーブラケット (3) セミコロン (4) Backspace と Delete キー (5) Escape キー (6) Enter キーの押下・入力回数の平均値と標準偏差を，学習者ごとに計算した．その結果を Welch の t 検定で検定した結果を **表 3** に示す．その結果，補完機能を伴う操作において，括弧とカーリーブラケットの入力回数が有意に少ないことと，Enter キーの押下回数が有意に多いことが確認できた．なお，多くの数値が 1 以下になっている理由は，文字の入力 1 回につき，3 個の操作イベントが記録されることと，編集途中の保存操作により，イベントが 100 個に満たない操作ログも多数含まれることに起因している．if 文や for 文を補完すると，括弧やカーリーブラケットの入力が省略できることが要因と考えられる．括弧やカーリーブラケットは，SHIFT キーを押しながら入力する必要があるため，タイピングスキルの低い学習者にとっては若干敷居が高い．そのため，これらのキー入力を軽減できることは，一定の意義があると考えられる．Enter キーの回数が増えている理由は不明だが，補完候補の確定操作や，ブロック部分（カーリーブラケットの内部）に行を追加する操作が影響している可能性がある．関連して，セミコロンの回数にも有意傾向がみられた．カーソルキーや Backspace，Escape キーの押下回数には，有意な差はみられなかった．確認・検証はしていないが，初学者はカーソル移動をマウス操作によって行うことが多いのではないかと考えられる．

3.2 アンケートの結果と分析

中間試験の直後に，アンケート調査を行った．アンケート調査の目的は，試験問題の難易度や時間設定に対する学生の意識，エディタの支援機能に対する学生の意識と利用の有無，ならびに学生のスコアについての関連を明らかにすることである．アンケート質問項目について，共通する選択肢でまとめたものを，以下の箇条書きに示す．なお，「字下げ調整機能」とは，図 1 の上部中央のボタンによって，字下げを一括調整する機能であり，講義の第 1 回目から提供されていた．

- 質問 1-1 / 2-1 / 3-1：中間試験の 1 / 2 / 3 問目を解くのに，どれくらい時間がかかりましたか？
  (1) 5 分程度 (2) 10 分程度 (3) 15 分程度 (4) 20 分程度 (5) 25 分程度 (6) 30 分程度 (7) 30 分以上 (8) 解けなかった（もし十分な時間があれば，解けたとおもう）(9) 解けなかった（十分な時間があっても，解けなかったとおもう）
- 質問 1-2 / 2-2 / 3-2：あなたは，中間試験の 1 / 2 / 3 問目の問題のレベルについて，どう感じましたか？
  (1) とても簡単だった (2) どちらかというと簡単だった (3) どちらともいえない (4) まあ難しかった (5) とても難しかった
- 質問 4：あなたの現在の yatt （タイピング練習ソフト）の最高スコアは何点くらいですか？
  (1) 1–60 点 (2) 61–80 点 (3) 81–100 点 (4) 101–120 点 (5) 121–140 点 (6) 141–160 点 (7) 161 点以上
- 質問 5：ソースコードの編集中に，「字下げ調整機能」は使いましたか？
- 質問 7：ソースコードの編集中に，「Tab キーによる補完機能」は使いましたか？
  (1) 存在を知らなかった・気づかなかった (2) 知っていたが，使わなかった (3) たまに気がついたときに使った (4) 積極的に使った
- 質問 6：「字下げ調整機能」は，あなたにとって，どの程度必要ですか？
- 質問 8：「Tab キーによる補完機能」は，あなたにとって，どの程度必要ですか？
  (1) 全く必要ない (2) あまり必要ない (3) あってもなくてもよい (4) できれば必要 (5) とても必要

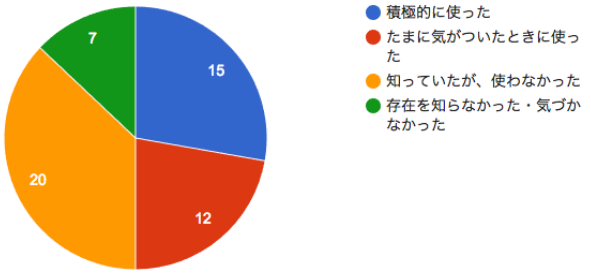

図 4 質問 7：「補完機能の使用頻度」の回答集計

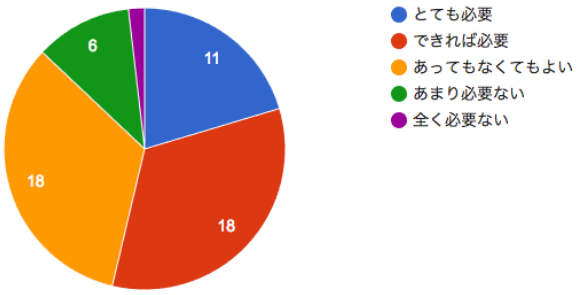

図 5　質問 8：「補完機能の必要性」の回答集計

有効回答は 54 件であった．**図 4** と**図 5** に，それぞれ補完機能の使用頻度（質問 7）と必要性（質問 9）の回答結果を示す．約半数の学習者が，補完機能を使用していたことと，必要性を感じていたことが伺える．

アンケート結果と，中間試験までのスコアの相関係数，および，無相関の検定を行った結果を**図 6** に示す．水色**は 1%有意，緑色*は 5%有意，黄色無印は 10%有意傾向を示している．図 6 の結果から，学習者が感じる問題の難易度と，回答にかかる時間，および，スコアとの相関の高さが確認できた．また，自動補完および字下げ調整機能について，使用頻度と必要性のあいだの相関の高さも確認できた．なお，問題 2 の時間と，字下げの必要性に，負の相関がみられる．これは問題 2 に時間をかけた学習者ほど，字下げ機能の必要性を感じていないという解釈ができる．問題 2 は for 文を使うと短時間で解ける問題であることから，for 文の理解や活用ができていない学習者は，字下げ機能の有用性や利便性を見出せていないということを示している．

また，今回の講義の中間試験までの結果ではあるが，スコアとタイピングスキルの相関は，有意な差が確認できなかった ($r = 0.21$, $t(52) = 1.56$, $p = 0.12$)．これは 1. で述べたように，タイピングスキルが成績に与える影響は低い方が望ましいと考えている我々の立場からみると，悪くはない結果といえる．

今回の試験においては，難易度が高いと感じる学習者が半数以上と多く，自由記述コメントからもタイピング以前の理解不足や練習不足を感じている学習者が多かった．また，今回の実践においては，講義の最初から提供している字下げ調整機能についても，利用者や，必要性を感じている学習者の割合が半数程度であった．そのため，テキスト記述において読みやすく，メンテナンスしやすいコードを書くことの重要性や，その意義や効果の理解まで到達していない学習者が一定数いたと考えられる．このような，テキスト記述に対する意識や動機付けがまだなされていない段階の学習者に対しては，Blockly Processing[12] のような，ブロック記述によるプログラミング環境が適している可能性がある．

4.　関連研究

キーボードによるタイピングスキルの影響を軽減しつつ，不慣れな文法による入力ミスを避けるため，初学者用プログラミング環境においてブロック型の入力を導入する研究や，入力補助機能を備えたプログラミング学習環境の研究が行われている．松澤らは，ブロック記述方式とテキスト記述方式を相互に変換できる環境を構築し，Java 言語による演習実践を行っている [13]．実践の結果，プログラミング学習の進行に沿って，ブロック記述からテキスト記述への緩やかに移行していくことが確認された．この研究は，初学者を対象としたプログラミング演習において，長期的な実践と調査を行っている点で本研究との類似性がある．ただし，テキスト記述方式における，自動補完機能についての言及や，その効果の調査はされていない．

初学者用プログラミング学習環境 PEN では，プログラム入力支援ボタンによって，学習者が条件文や繰り返しといった制御構造や，関数を簡単に入力できる補助機能を備えている [14]．日本語を含む拡張 DNCL の構文を正確かつ短時間で入力するには必要不可欠な機能であると考えられる．画面上にメニューボタンが常時表示されているため，自動補完機能よりも気付きやすく，またメニューの表現も初学者にわかりやすくできる点は，自動補完より優れているといえる．その反面，ボタン表示に画面が占有されてしまうことや，メニュー選択の際にはマウス操作が必要になるため，キー入力を継続しにくい点は若干の欠点といえる．

市村ら [1] は，プログラミング初学者が抱える問題を早期発見することを目的として，CodeMirror エディタを利用した Web IDE を導入し，操作ログ・エラーログの収集と解析を行っている．Web IDE を利用したプログラミング学習環境による実践研究という観点において，本研究と関連している．本研究ではコード補完機能の導入により，初学者に積極的に部分コードの入力支援を促している点と，Java 言語ではなく Processing 言語を対象としている点が異なる．

初学者の学習に限らずに，ソフトウェア開発者の生産性や利便性を高めるための入力・参照インタフェースの改良や，フレームワークについての研究もなされている．Ko らは，テキスト記述中心のエディタを拡張し，柔軟な表現を導入するためのフレームワークを提案している [15]．Brandt らは，Web 検索インタフェースを IDE に組み込む方法を提案している [16]．Oney らも，サンプルコードを効果的に検索，挿入できるエディタ Codelets を開発し，jQuery Mobile を用いた Web 開発を対象とした評価実験を行っている [17]．薄羽らは，自然言語によるコメントを入力すると，関連する候補を推薦する手法を提案し，高校生を対象とした評価実験を行っている [18]．本研究ではシンプルな

	Q1時間	Q1難易	Q2時間	Q2難易	Q3時間	Q3難易	タイピング	字下使用	字下必要	補完使用	補完必要	スコア
Q1時間		0.59 **	0.47 **	0.32 *	0.37 **	0.24	−0.04	−0.03	−0.12	−0.03	−0.14	−0.49 **
Q1難易	0.59 **		0.26	0.60 **	0.15	0.59 **	−0.22	0.04	−0.11	0.02	0.06	−0.57 **
Q2時間	0.47 **	0.26		0.64 **	0.48 **	0.13	−0.17	−0.11	−0.38 **	−0.00	0.24	−0.37 **
Q2難易	0.32 *	0.60 **	0.64 **		0.15	0.42 **	−0.26	−0.12	−0.26	−0.12	0.15	−0.62 **
Q3時間	0.37 **	0.15	0.48 **	0.15		0.44 **	−0.03	0.07	0.05	0.02	0.15	−0.23
Q3難易	0.24	0.59 **	0.13	0.42 **	0.44 **		−0.09	−0.03	0.06	−0.08	0.08	−0.35 *
タイピング	−0.04	−0.22	−0.17	−0.26	−0.03	−0.09		−0.04	0.03	−0.00	−0.16	0.21
字下使用	−0.03	0.04	−0.11	−0.12	0.07	−0.03	−0.04		0.50 **	−0.04	−0.08	0.07
字下必要	−0.12	−0.11	−0.38 **	−0.26	0.05	0.06	0.03	0.50 **		0.04	0.03	0.14
補完使用	−0.03	0.02	−0.00	−0.12	0.02	−0.08	−0.00	−0.04	0.04		0.70 **	−0.02
補完必要	−0.14	0.06	0.24	0.15	0.15	0.08	−0.16	−0.08	0.03	0.70 **		−0.11
スコア	−0.49 **	−0.57 **	−0.37 **	−0.62 **	−0.23	−0.35 *	0.21	0.07	0.14	−0.02	−0.11	

図 6 アンケート結果およびスコアの相関 (**は 1%有意，*は 5%有意，黄色は 10%有意傾向)

コード補完機能の導入に関して，初学者の活動を 3 週間程度にわたって記録し，キー入力イベントの統計や試験の成績との関連を調査している．

一般的に普及・利用されている統合開発環境の機能に関する調査として，Satav らは，C/C++, Java 言語用の統合開発環境の比較を行っている [19]．自動補完機能についても，C/C++環境と Java 環境に分けて，機能が利用できるかどうかを IDE ごとに羅列している．

5. おわりに

Web IDE は，ブラウザのみで演習環境が利用できるため，初学者が気軽に演習に取り組めるという点で優れている．Processing 言語を対象とした Web IDE の利便性を高めるため，我々は自動補完機能を導入した．

約 3 週間の講義での試用の結果，自動補完機能を活用する学習者とそうでない学習者に二分された．利用頻度の高い補完候補は if 文や for 文のほか，画面描画関連の関数呼び出しであった．if 文や for 文のスニペットには，括弧やカーリーブラケットを含んでいたため，補完機能を利用した編集行為の前後は，利用しなかった編集行為の前後と比較して，括弧やカーリーブラケットの入力数が有意に低かった．中間試験の直後のアンケートから，補完機能は約半数の学習者が利用していた．また，補完機能の使用頻度と，学習者が感じる必要性には，強い正の相関が認められた．

今回は 3 週間程度と短期間の利用であったことや，[Tab] キーによる明示的な操作を必要としたことから，学習者が自動補完を利用する動機付けが弱かったり，日常的な利用に結びついていない面があった．今後は，長期的な利用や習慣づけによる効果を検証していきたい．

自動補完という概念および機能は，初学者の期間だけでなくプログラミングに習熟・上達してからも継続的に活用でき，また有用性も高い．そのため，初学者のうちから自動補完機能の習慣的な利用を推奨することは，長期的なプログラミングスキルの涵養としても意義があると考えている．

参考文献

[1] 市村哲, 梶並知記, 平野洋行．プログラミング演習授業における学習状況把握支援の試み．情報処理学会論文誌, Vol. 54, No. 12, pp. 2518–2527, 2013.

[2] Max Goldman, Greg Little, and Robert C Miller. Real-time collaborative coding in a web IDE. In *Proceedings of the 24th annual ACM symposium on User interface software and technology*, pp. 155–164. ACM, 2011.

[3] Vu Nguyen, Hai H Dang, Kha N Do, and Thu D Tran. Learning and Practicing Object-Oriented Programming Using a Collaborative Web-based IDE. In *Frontiers in Education Conference (FIE)*, pp. 1–9. IEEE, 2014.

[4] 長島和平, 長慎也．Tonyu System 2 ゲーム制作を通じたプログラミング学習に適したフレームワーク．情報処理学会研究報告．コンピュータと教育研究会報告, Vol. 2015, No. 2, pp. 1–8, March 2015.

[5] 長島和平, 長慎也, 間辺広樹, 兼宗進, 並木美太郎．Web ブラウザを用いたプログラミング学習支援環境 Bit Arrow の設計と評価．研究報告コンピュータと教育 (CE), Vol. 2017, No. 2, pp. 1–8, February 2017.

[6] 三浦元喜．Processing Web IDE を用いたプログラミング基礎教育の試み．情報処理学会情報教育シンポジウム (SSS2013) 予稿集, pp. 225–231, August 2013.

[7] Hiroshi Kimura. Yatt: Yet Another Typing Trainer. https://literacy.melt.kyutech.ac.jp/yatt.html (2018 年 7 月 5 日確認).

[8] Mitchel Resnick, John Maloney, Andrés Monroy-Hernández, Natalie Rusk, Evelyn Eastmond, Karen Brennan, Amon Millner, Eric Rosenbaum, Jay Silver, Brian Silverman, et al. Scratch: programming for all. *Communications of the ACM*, Vol. 52, No. 11, pp. 60–67, 2009.

[9] Neil Fraser, et al. Blockly: A visual programming editor. *URL: https://developers.google.com/blockly/*, 2013.

[10] Neil Fraser. Ten Things We've Learned from Blockly. In *2015 IEEE Blocks and Beyond Workshop (Blocks and Beyond)*, pp. 49–50, October 2015.

[11] 三重野剛, 三浦元喜．プログラミング演習におけるブロック型インタフェースの導入と実践．大学 e ラーニング協議

会/日本リメディアル教育学会合同フォーラム 2017, pp. 16–17, February 2018.
[12] 高知工科大学高田研究室. Blockly processing editor. https://ytakata69.github.io/blockly-processing/ (2018 年 6 月 3 日確認).
[13] 松澤芳昭, 保井元, 杉浦学, 酒井三四郎. ビジュアル-Java 相互変換によるシームレスな言語移行を指向したプログラミング学習環境の提案と評価. 情報処理学会論文誌, Vol. 55, No. 1, pp. 57–71, January 2014.
[14] 西田知博, 原田章, 中村亮太, 宮本友介, 松浦敏雄. 初学者用プログラミング学習環境 PEN の実装と評価. 情報処理学会論文誌, Vol. 48, No. 8, pp. 2736–2747, August 2007.
[15] Andrew J. Ko and Brad A. Myers. Barista: An implementation framework for enabling new tools, interaction techniques and views in code editors. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 387–396. ACM, 2006.
[16] Joel Brandt, Mira Dontcheva, Marcos Weskamp, and Scott R Klemmer. Example-Centric Programming: Integrating Web Search into the Development Environment. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 513–522. ACM, 2010.
[17] Stephen Oney and Joel Brandt. Codelets: linking interactive documentation and example code in the editor. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pp. 2697–2706. ACM, 2012.
[18] 薄羽大樹, 宮下芳明. コメントイン: コメントを先に書くことによる新形態の API リファレンス. 第 23 回インタラクティブシステムとソフトウェアに関するワークショップ (WISS2015) 論文集, pp. 67–72. 日本ソフトウェア科学会, November 2015.
[19] Sampada K. Satav, S. K. Satpathy, and K. J. Satao. A Comparative Study and Critical Analysis of Various Integrated Development Environments of C, C++, and Java Languages for Optimum Development. *Universal Journal of Applied Computer Science and Technology*, Vol. 1, pp. 9–15, January 2011.

付　　録

A.1 補完機能で利用した Processing キーワード・スニペットの一覧

注：\n は改行を示す.

```
void
setup(){\n}
size(w,h);
background(200);
fill(,,);
ellipse(cx,cy,w,h);
noFill();
strokeWeight(3);
stroke(,,);
rect(x,y,w,h);
textSize(50);
width
height
mouseX
mouseY
int
float
textAlign(CENTER);
text("",width/2,height/2);
text("",x,y);
println();
if () {\n  } else {\n  }
else
for ( int k=0 ; k<10 ; k++ ) {\n  }
while () {\n  }
mousePressed(){\n}
mouseClicked(){\n}
mouseMoved(){\n}
keyPressed(){\n}
keyCode
frameRate(20);
line(x1,y1,x2,y2);
triangle(x1,y1,x2,y2,x3,y3);
quad(x1,y1,x2,y2,x3,y3,x4,y4);
noStroke();
mousePressed
mouseButton
PImage
loadImage("http://");
draw(){\n}
image(img, x, y, img.width, img.height);
class
new
random(low,high);
floor();
ceil();
round();
LEFT
CENTER
RIGHT
pushMatrix();
popMatrix();
translate(dx,dy);
rotate(radians(90));
colorMode(HSB,255);
textFont(createFont("Arial",20));
ArrayList list;
list = new ArrayList();
list.add();
list.size();
list.get(0);
list.remove(0);
return ;
mp3("game01/jump12");
```

A.2 講義開始時点での自己申告タイピングスコアの分布

表 A·1 自己申告タイピングスコアの分布 (数字は人数)

数値	1	2	3	4	5	6	7
スコア	1〜	61〜	81〜	101〜	121〜	141〜	161
の範囲	60	80	100	120	140	160	〜
2018	26	19	10	7	4	0	0
2017	23	12	6	1	1	1	1
2016	4	3	5	27	16	5	9
2015	7	9	11	15	11	5	14
2014 (1)	19	16	18	3	3	8	8
2014 (2)	13	16	12	5	0	0	3
2014 (3)	14	15	12	5	1	1	4

A.3 中間試験の問題

図 A·1 中間試験の問題 (動作版： `http://miura.tobata.asia/kiso/index.html#exam201801.md`)

A.4 自動補完機能の利用回数

表 A·2 補完機能で選択されたスニペットの利用回数

選択回数	選択スニペット
123	`if () {\n } else {\n }`
63	`rect(x,y,w,h);`
62	`for ( int k=0 ; k<10 ; k++ ) {\n }`
51	`background(200);`
49	`int`
47	`ellipse(cx,cy,w,h);`
42	`fill(,,);`
26	`mouseX`
24	`mousePressed(){\n }`
24	`frameRate(20);`
24	`void`
21	`else`
16	`random(low,high);`
15	`text("",x,y);`
15	`float`
14	`mouseClicked(){\n }`
12	`draw(){\n }`
7	`mouseY`
6	`text("",width/2,height/2);`
5	`line(x1,y1,x2,y2);`
5	`stroke(,,);`