Vol.2020-HCI-189 No.1
2020/9/7

typing.run: 初学者のプログラミング学習を支援するプログラムタイピングシステムの提案と実践

又吉康綱[†1]　中村聡史[†1]

概要：初学者がプログラミング学習においてつまずく点は様々であるが，著者らのこれまでの数年にわたる大学初年次のプログラミング教育の経験では，タイピング速度の遅さや，プログラミングに利用される英単語，カッコやセミコロンなど特殊文字の入力に抵抗があることが妨げとなっていることが観察されていた．そこで本研究では，プログラミング学習を円滑に進めるため，プログラムの逐次的な実行などによってプログラムの動作を把握しつつ，自身や他者と競いながらタイピング速度を上げつつプログラムを学ぶ，プログラムタイピングシステム typing.run を提案および実装した．また，本システムを演習型プログラミング授業にて，授業前までの課題として毎授業ごとに複数回学生に取り組むことを課した運用の結果から，タイピング速度にばらつきがなくなっていること，全体的なタイピング速度が向上していること，またアンケートからプログラミング習得に効果的だったことなどが明らかになった．

キーワード：タイピング練習，プログラミング学習，オンライン講義，COVID-19

1. はじめに

大学の初年次プログラミング教育では，初学者から経験がある学生まで含めて広くカバーする必要があるため，各大学，各学部で教員が趣向を凝らし，実施している．ここで,一般的に大学における必修のプログラミング教育では，主に初学者の底上げが重要であり，また教員と TA（ティーチングアシスタント）が，多様なレベルの数多くの学生を担当する必要があるため工夫が必要となる．

我々が所属する明治大学総合数理学部先端メディアサイエンス学科でも，学部 1 年生の必修講義としてプログラミングの演習講義が行われており，100 人以上の学生が一斉に学んでいる．先述の通り，学生のプログラミングに対する理解度はひとそれぞれであり，そもそもコンピュータに対して不慣れであったり，タイピングが不慣れであったりするなど，その苦手度合いも多様である．その結果，限られた時間内で，プログラミングが苦手な学生の理解度を如何に向上させるかは,教育側にとって大きな課題である．

我々はこれまで，講義教材の開発や課題の開発[1]，予習のためのドリルの開発[2]，小テストの実施，発表会の実施など様々な工夫を行ってきたが,十分であるとは言い難かった．また，これまでの数年にわたる講義での経験から，プログラミングが苦手な学生の理解度が向上しない根本的な原因に，タイピングの技量と基本的な命令の理解が不足していることが観察されていた．

ここで Python や C，Java や Processing [1]など，大学の初年次教育で採用されるプログラミング言語では，多くの日本人学生にとって日常生活では使用しないアルファベットを含む英単語や，カッコやセミコロンなどの特殊記号をタイプする必要があり,このことが学びの障壁となっている．タイピング速度を向上させるには，実際に手を動かして練習することが重要であり，世の中には様々なタイピングに関するサービスやゲームなどが提供されているが，大半のシステムは，日本語のローマ字入力や，英単語の入力を対象としていることが多い．そのため，こうしたタイピングシステムを利用してある程度タイピング速度を向上させたとしても，プログラミングにおいて多用される記号の入力などで時間がかかってしまい，結果としてプログラムのタイピングに苦戦する学生が多数存在していた．こうした学生は，プログラムを書き写すのにも多くの時間が掛かってしまい，学習や課題解決が困難になっていた．

また，プログラミング学習においては，プログラミングの際によく使う基本命令や関数への理解と，そうした関数へ慣れ親しむことも重要となる．学び始めのときは，使う関数が少ないため把握することは難しくないが，講義の回数を重ねるごとに新しい関数が登場するばかりか，条件分岐や繰り返し，配列，関数の定義，クラスなど，使い方を覚えて理解する対象が増えてくる．そのため，学習の過程で理解が不十分であると，次に進むことができなくなり，学習が止まってしまう．また，使いたい関数を思い出せずに，リファレンスや検索サイトなどで毎回調べる必要があるなど，わからないことが積み重なることも，学習をスムーズに進めるための妨げになっている．

以上より，プログラミングが苦手な学生の理解度向上には，英単語や記号などのタイピング練習による入力速度の向上と，基本命令や関数に繰り返し触れることで慣れて調べなくても自力で思い出し利用できるような反復による記憶が必要となる．

そこで本研究では，プログラミングを学び始める大学の学部 1 年生に対して，プログラミング特有のタイピングの

†1 明治大学
Meiji University

[1] http://nkmr.io/lecture/
[2] http://drill.nkmr.io/

練習に反復的に取り組んでもらい，またよく使う基本命令の定着を目指したプログラミング学習を円滑に進めるためのプログラムタイピングシステム typing.run を提案および実装する．ここで，単純にプログラムを写経する場合，今入力している文字列が一体何のためであり，何をやっているのかを把握できずに身につかないと考えられる．また，ひとり黙々とタイピング練習するのは，退屈であると考えられる．そこで本システムでは，タイピング不要だが理解を促すコメントをタイピング対象のプログラム中に付与するとともに，1 行ごとにプログラムを逐次実行させることで書いた関数の挙動の理解を促進するための工夫や，ユーザ同士でタイピングスコアを競い合えることでやる気を高められるような仕組みを構築した．

また本研究では，システムを実際のプログラミングの全 11 回の演習講義において課題として設定し，受講生に複数回実施してもらった．その運用に基づく分析結果をもとに，システムの有用性について考察を行う．

2. 関連研究

プログラミングのパフォーマンスとタイピング速度の間に相関があることを，Thomas ら[2]は複数の実験結果から示している．また，プログラミングの技量をタイピングのキーストロークの時間間隔から求める研究を Leinonen ら[3]が行っており，精度は十分ではないものの技量を時間間隔から求めることができる可能性を示唆している．これらより，プログラミングにとってタイピングが重要であることが言え，本研究で単純にタイピング速度を向上できるだけでも，プログラミングの上達につながると期待される．

プログラミング言語のタイピングと，大学での演習授業に着目して行われた研究に中田[4]の研究が挙げられる．この研究では，C 言語のプログラムを用いて授業前に毎回タイピング練習することによって，成績やタイピング速度との関係性を複数年に渡って調査している．その結果，タイピングの実施自体に成績向上は見られないものの，成績とタイピング速度に一定の相関があることを明らかにしている．また，喜多ら[5]は，写経型プログラミングを提唱しており，自著の本[6]を使った反転授業を行った報告をしている．ここでは，学生に本の内容に沿って授業時間外に C 言語のプログラムを写経させて，その進捗を提出させている．また授業時間内では，クラス全員の進捗をグラフで示して学習状況を比較させている．この授業スタイルを行った結果，最終課題の提出物から一定の成果を得ることができたとしている．これらの研究の目的は本研究と類似しているが，関数などのプログラムの理解を促す仕組みの点で異なっている．また本研究では，プログラミングのタイピング練習と同時に，そのプログラム自体の意味や挙動を理解させることに重きをおいている．

一方，日本語でのローマ字入力や英単語のタイピング練習を大学での演習授業で実施した吉長ら[7]は，内発的動機づけにより，学生にタッチタイピング練習を継続的に 1 年間行わせた．その結果，時系列での打鍵データより習熟のパターンに 11 種類あることを示している．

大学におけるプログラミング演習型講義の支援に関する研究として我々[8]は，これまでに TA と学生の抽象的思考に着目した指導を楽にするための手法を提案し支援を行ってきた．しかし，これまでの研究は講義中の TA と学生を支援するものであり，その事前準備には不十分であった．

こうした研究とは異なり，本研究は，タイピング速度の向上を踏まえつつ，プログラミングの学習も行っていくことで，初学者のプログラミング学習を支援するものである．

3. 提案手法

プログラミングを学び始めた初学者が，モチベーションを保ったままタイピング練習やプログラムへの理解を深めるようにするには，内発的動機づけと外発的動機づけ，そして理解を促すための各種の仕組みが重要になる．そこで，プログラムタイピングシステムの実現にあたり，その理解を促す仕組みと，動機づけを行う仕組みについて述べる．

3.1 プログラムの理解を促す仕組み

プログラミングを理解するためには，ただ提示されたプログラムをタイピングするのではなく，命令を理解しながらタイピングする必要がある．ここで，タイピング対象となる問題のプログラムをただ提示するだけでは，理解にはつながらない．事前にその文字列が何を意味しており，その入力がどういった影響を及ぼすのかをユーザが学習してくれればよいが，そうしたことを多くのユーザ（講義においては学生）に要求するのは難しい．そのため，自身が入力している文字列が何を意味し，その結果がどのような影響を及ぼすかをそれとなく理解させる仕組みが重要になる．

そこで，ユーザがタイピングしている行の文字列（プログラム）の 1 行上に，タイピング対象とはしない説明コメントを提示し，知識として理解を促す．ここで，コメント行は色を変えるとともに，前のプログラムを入力した後にコメント行に来た場合には，自動的にコメント行の後の行にスキップすることでコメントを無視可能とする．

また，ユーザの入力に応じて徐々にプログラムを動作させることによって（入力途中の場合は，閉じカッコがないなどプログラムは本来動作しないが，内部的に補完することによって実現），ウィンドウ作成にまつわる行を入力するとウィンドウが現れ，背景指定にまつわる行を入力すると背景の色が変わり，円を描画する行を入力すると円が描画されるといったように，内容を視覚的に把握可能とする．ここでは，タイピング対象の問題にも工夫を行うことによって，1〜2 行の入力で視覚的情報が徐々に変化していくも

のとする．

一方，プログラミングにおいて利用される関数には，様々な引数をもつものが多い．ここで，タイピング練習では，その引数として与えられる値が固定されているため，最後まで入力したとしても，タイピング用に指定された引数の時の挙動しか見ることができない．このような試行錯誤ができないことは，プログラミング習得においてはデメリットである．そこで，タイピング練習後にそのタイピングしたプログラムを編集し，円の大きさや色を変更したり，表示順番を変更したりするといったように，関数の引数や記述の順番などを気軽に変更可能とすることにより，理解を促す．

3.2 内発的動機づけと外発的動機づけ

プログラミングに限らず，タイピング練習は繰り返し行うことが重要である．そのためには，内発的動機づけと，外発的動機づけを考慮する必要がある．

まず，内発的動機づけについては，タイピングのスコアを Character per Minute（CPM）として，1 分間にタイプできた文字の数を算出することで，各問題における自分のタイピング速度の指標として見ることができるようにし，少しでも良い自己ベストの CPM を目指せるようにする．また，CPM ではタイピング速度を早めることに注目しているため，タイピングの正確性を高めてもらうために，タイピングの打ち間違えたミスの数に関する情報も提示する．

一方，外発的動機づけとして，多くのタイピングシステムでは他者のスコアを提示したり，ランキングを提示したりといった工夫をしている．そこで，本システムでも同級生の CPM と自分の CPM とを比べることができるようにすることで競争心を煽り，モチベーションを維持できるようにする．ここで，ランキングはある程度興味をもったユーザに対しては効果的ではあるものの，もともとタイピング速度が遅いようなユーザにとっては，負の効果になり，まったく取り組まない可能性もある．そこで，講義の予習のノルマとしてタイピング回数を取り入れることで，最低限のタイピングを実施してもらい，そこから興味をもったユーザにランキングなどへの挑戦を促す．

4. typing.run

我々の所属する学科のみならず，多くの大学の初年次プログラミング講義で採用されている Processing [1]を対象とし，提案手法を用いたシステム typing.run[3]を実装した．

4.1 実装

システムは，誰もがアクセスしやすいように Web アプリケーションとして実装した．実装には JavaScript，MySQL 等を用いた．サーバーサイドは Node.js と MySQL 等で構築し，問題の配信やユーザのタイピング結果の保存などを行った．クライアントサイドは Processing.js [9]を利用しつつ，JavaScript で実装し，取り組む問題を選択するページとタイピングを行うページとを用意した．また，大学の認証システムと紐づけることにより，ユーザの識別を行った．

図 1　タイピングする行の上に入力しないコメントを表示し，カッコを内部補完しながらタイピングした行から実行される例

一方，プログラムの理解を促す仕組みにおいて説明した通り，各行端ごとにタイピングされたプログラムの入れ子構造が途中だった場合に，強制的に不足している個数分のカッコを内部的に挿入することで，各行の逐次実行を実現した．

図 1 は，実現したシステムを利用してタイピングを行い，そのタイピングした行に応じてプログラムが実行されている様子である．このプログラムでは，setup 関数内に書かれた size の行を入力後にウィンドウが作成され，次に background の行を入力後に背景が白色で描画され，ellipse

[3] https://typing.run

図 2　問題選択ページのスクリーンショット

の行を入力後に円が描画されるものとなっている．本来なら，setup 関数内のすべてをタイピングして setup のカッコを閉じるまで（「}」を入力するまで）描画されないが，内部処理的にカッコを補完することで逐次実行を行っている．

4.2　問題の開発

プログラミングについての知識がまったくない初学者に対して，いきなり繰り返しや配列といった知識が必要な問題を提示することは適切ではない．適切な学びを得るには，理解の度合いや講義の進度と合わせてタイピング課題に挑戦できるようにすることが重要であると考え，タイピング問題も講義に連動するようにセクションごとに分割した．

各問題の作成においては，講義の予習となるように，事前に配布している予習資料[4]にあるプログラムを積極的に採用するとともに，問題数を 4〜10 と適切になるように調整した．ここでは，プログラムをできるだけ意味のあるものとするため，決め打ちの値を入力するのではなく，変数名をわかりやすい単語などに変更した．また，あまりに複雑なプログラムや，入力行数が多いプログラム，表示結果が面白くないプログラムなどは採用しなかった．一方，プログラムを 1〜2 行入力するだけで徐々に提示されるようにするため，プログラムの記述順番などを工夫した．

なお，システムの逐次実行の都合上，while を使用すると終了条件を満たすためのプログラムが書かれる前に実行されてしまい，無限ループとなり応答不能になるため書き方に制限があった．そのため，提示するプログラムでは while の部分入力を行った際に無限ループにならないように，適切な処理を行った．この無限ループに関する制限については，今後改善を図っていく予定である．

上記を考慮して，プログラミング演習の 11 回分の講義に

図 3　タイピングページのスクリーンショット

対して，合計 83 個の問題を作成した．

4.3　利用方法

本システムは，大きく分けてセクションごとの問題を選択する問題選択ページと，実際にタイピングを行うタイピングページに分かれている．図 2 の問題選択ページでは，取り組む問題の選択や自己ベスト CPM の確認，セクションごとにランキングの確認，他者のハイスコア CPM とタイピングした回数と関係性を示した表などを確認することができる．また，取り組む問題のタイトルをクリックすることで，タイピングページに遷移される．

図 3 のタイピングページでは，左側にタイピングする問題の提示，右側に実行画面，下部にコンソールを設けている．ここで入力前の文字列はすべて灰色文字になっているが，タイピングしていくと予約語は青色になり，その他は白文字になるようにした．ユーザがタイピングしている文字を示すキャレットは，灰色のマーカに黒文字で表現した．なお，入力不要のコメントは緑色にした．

下側に並んだボタンでは，プログラムの編集や実行が可能なモードやクリップボードにソースコードをコピーできる機能が利用できる．また，実行画面の下ではタイピングを初めてからの経過時間，現在の CPM やタイプミスの回数などを確認することができる．

5.　typing.run の運用と分析

前章で述べた typing.run を 2020 年 5 月 6 日より公開し，第 2 著者が担当している明治大学総合数理学部先端メディアサイエンス学科の学部 1 年生向け必修授業であるプログラミング演習 1 で運用を行った．なお，運用期間は 2020 年 6 月 22 日から 2020 年 7 月 27 日までの毎週月曜日と水曜日であり，タイピング課題は講義開始前までのノルマとした．

受講生には，授業開始期間前の typing.run 公開直後から授業で使用することの周知を行い，練習やデバッグを兼ね

[4] http://nkmr.io/lecture/

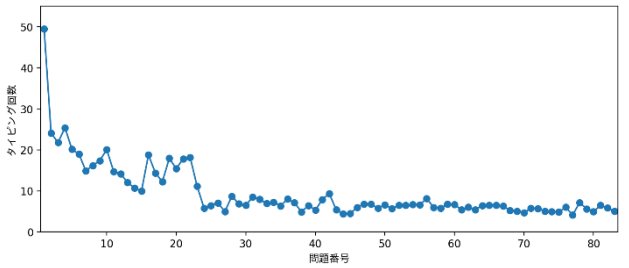

図 4　問題と取り組み回数の平均

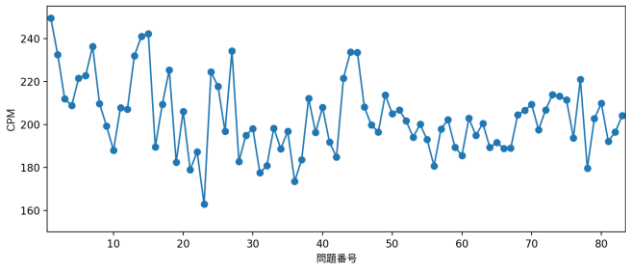

図 5　問題とハイスコア CPM の平均

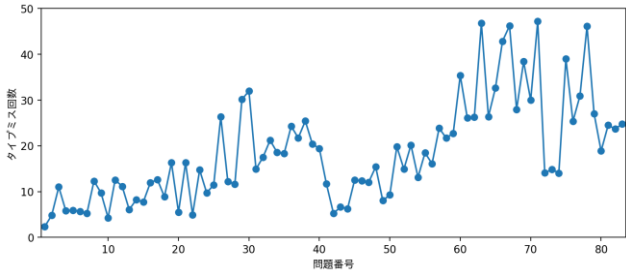

図 6　問題とタイプミス回数の平均

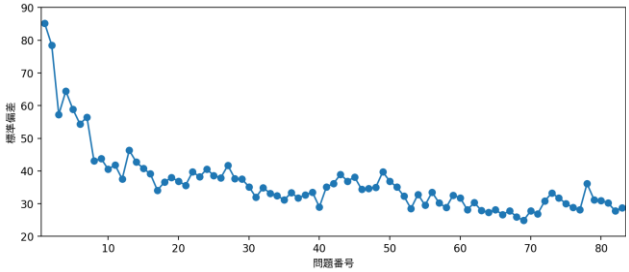

図 7　問題とハイスコア CPM の標準偏差

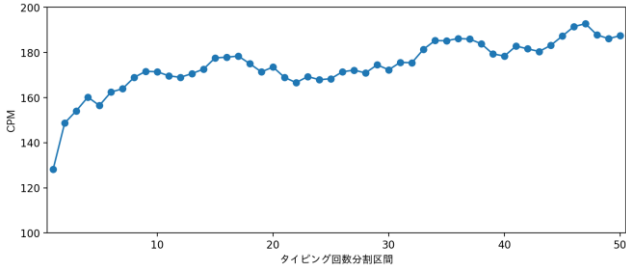

図 8　タイピング回数を系列順に 50 分割した区間と
それぞれの区間で全員の CPM の平均

ての利用を呼びかけた.

5.1　タイピング結果と分析

運用で収集したデータをセクションごとの授業開始前までにタイピングされた 118 人分，タイピング 88,293 回分のデータを集計した．なお，タイピングにおいてマクロ等を使った不正行為が疑われるデータや，タイピング時間が 30 分を超えていて放置されたと思われるデータは分析から除外した．結果的に，88,055 回分のデータを分析対象とした.

ここで，全 83 問のタイピング問題を横軸にとり，指標ごとに折れ線グラフで比較した結果を図 4〜7 に示す.

図 4 は, 問題ごとのタイピング回数の変化を示しており，初めの方は 20 回以上タイピングに取り組んでいたが，25 問目を過ぎた辺りから 5 回程度になっていることが分かる．1 問目の回数がかなり多いのは，プログラムが短く，速度を競うために何度タイピングした受講生が多いためである.

図 5 は，問題ごとの各受講生の最速 CPM を平均したものである．全体を通して CPM の平均値が 200CPM 付近になった問題が多かったことがわかるが，この結果だけを見ると全体として入力速度に変化はない.

図 6 は，問題ごとに最速 CPM だった時のタイプミス平均を算出したものである．この結果から，後半の問題になればなるほどタイプミスが増えていることがわかる．これは，問題の文字数が増え，カッコやセミコロンなど複雑な表記が増え，結果的に難易度が上昇したことが原因であると考えられる.

図 7 は，問題ごとの最速 CPM の標準偏差の推移を示したものである．この結果より，受講生のタイピングの速さに差がなくなっていることがわかる.

図 8 は, 各受講生がタイピングしたすべての回数を 50 分割し，その分割された区間ごとの CPM の平均を求めたものである．図 5 ではタイピング速度の伸びは観測されなかったが，図 8 の結果より，後半になるにしたがってタイピング問題が難しくなったにも関わらず, 全体として CPM の平均が上昇していることがわかる.

5.2　アンケート結果と分析

講義の第 11 回目の開始時（タイピング課題に取り組んでもらった中では最終回）に，受講生に対して typing.run に関するアンケートを実施し，回答してもらった．集まった 111 名分を集計したアンケート結果を表 1 に示す．なお，アンケートでは 5 段階（-2:まったく当てはまらない〜2:とても当てはまる）で評価してもらった.

表 1 の Q1 と Q2 の比較により，使用前と使用後のタイピングの得意度の変化を見ることができる. この結果より，使用後に評価が高くなっていることが分かる．また，提案手法に関する Q3，Q4，Q6，Q7，Q8 でも評価の平均値が正の値になった．しかし，Q5 のプログラムの変更ができる機能に関する評価の平均値は，負の値になった.

次に, 受講生ごとに Q2 で回答した評価値から Q1 で回答した評価値を引くことで，どれぐらいの段階数でタイピングが得意になったかを計算した結果を表 2 に示す．この結

表 1 アンケート項目とその分布と平均

質問項目		評価値の分布					平均
		-2	-1	0	1	2	
Q1	大学入学時点でタイピングが得意でしたか？	58	26	11	14	2	-1.12
Q2	授業終了後時点でタイピングが得意ですか？	11	24	35	35	6	0.01
Q3	同級生間のランキングや分布のグラフは意識しましたか？	12	36	11	43	9	0.01
Q4	1 行タイピングするごとに実行されることに意識しましたか？	8	24	15	42	22	0.41
Q5	プログラムを変更できる機能を使用しましたか？	43	36	6	21	5	-0.82
Q6	タイピング中にプログラムの内容をどのぐらい理解していましたか？	6	26	14	55	10	0.33
Q7	利用することで関数名を覚えることができましたか？	2	5	11	51	42	1.14
Q8	新しいことを学ぶ時に今後も利用したいですか？	4	2	13	44	48	1.17

果より，1 段階以上あがった受講生が約 71%となっていたことがわかる．

また，アンケートでは，システム使用以前のプログラミング経験の有無も聞いていた．そこで，システム利用のアンケート結果がプログラミング経験の有無によって違いがあったかを，Q1 から Q8 の質問項目ごとに評価値の平均を求めたものが表 3 である．この結果より，Q1 や Q2，Q6 では両者の間に大きな差があったことがわかる．

次に, typing.run を利用して身についたことや身につかなかったことなどの，ポジティブな意見とネガティブな意見を自由記述によって収集した．

ポジティブな意見では，「タッチタイピングができるようになった」「記号のタイピングが早くなった」などのタイピングに関するものや,「予習資料の理解が深くなった」「関数の働きがわかりやすかった」「コメントがわかりやすかった」などのプログラムの理解に関するもの,「自然に手が関数を覚えた」「ランキング表示で意欲が湧いた」「実装のアイディアが吸収できた」「予習のハードルが下がった」などの意見があり，65 件集まっていた．

一方，ネガティブな意見では,「スペースや記号を自分の好きなタイミングで打てない」といったプログラミング上級者からの意見や,「タイプミスを気にしない癖がついた」「スピード重視になって内容理解をおろそかにしてしまった」「タイピングが速いとプログラムができる気になってしまった」などのタイピングシステムであるからこその問題に関する意見が 45 件集まっていた．

表 2 タイピングの得意に関する評価値の差の件数（人）

	-4	-3	-2	-1	0	1	2	3	4
Q2-Q1	0	0	0	6	26	40	27	11	1

表 3 プログラミング経験有無による質問の評価値の平均

	経験なし(89 名)	経験あり(22 名)
Q1	-1.39	0.00
Q2	-0.12	0.55
Q3	0.01	0.00
Q4	0.39	0.50
Q5	-0.90	-0.50
Q6	0.12	1.18
Q7	1.12	1.18
Q8	1.20	1.05

6. 考察

6.1 タイピング結果からの考察

図 4 より，序盤の問題では，タイピングに取り組む回数が多かったが，次第に減っていることがわかる．これは，序盤の問題ではプログラムが短いため，短時間で終わることもあり,受講生が何回も取り組んだからだと考えられる．また，後半に一定の取り組み回数に落ち着いたのは，予習のノルマ回数分だけ取り組んでいたためだと考えられる．

全 83 個の問題では，プログラムの長さが異なっていたり，記号などの複雑なタイピングの多さが異なっていたりなど，問題自体の難易度さが異なるため，図 5 や 6 では，グラフの乱高下が激しくなっている．しかし，図 7 より受講生内でのスコアの分布の広がりを示す標準偏差がしだいに下がり，上位層と下位層のタイピング速さの差が縮まったと言え，下位層を底上げしていることが示唆される．

また，図 8 より実施回数が増えるにつれ，CPM の平均値が伸びていることが確認できた．問題の難易度が上がっているのにも関わらず，タイピング速度が向上していることから，全受講生のタイピング速度が次第に早くなるなど成長していると言える．

6.2 アンケート結果からの考察

タイピングの得意に関する Q1 と Q2 より, システムを使

ったことで,得意になった受講生が多くいたことが分かる．また，提案手法で述べた，外発的要因の意識が Q3 に，プログラムの理解促進が Q4 に対応し，それぞれ評価値の平均値が正の値になったことは，提案手法が有効である傾向が出ていると言える．一方，Q5 のプログラムの変更機能を使った受講生は少なかった．これは，そもそもその機能に気づいていないことや，講義の予習で変更する時間的な余裕がなかったことなどが原因と考えられる．今後の継続利用に関する Q8 についても高い評価値になったことに加えて，自由記述でもポジティブな意見が多く集まったことから，システム全体的な満足度が高いことが分かる．なお，自由記述におけるネガティブな意見の大半は，写経することに重点を置いたことでの弊害であるといえる．しかし，一部意見は実装を工夫することで解決することができるため，改善に取り組んでいく予定である．

表 2 の結果より，システムを使う前よりも後の方が，評価値が下がった受講生が 6 名いた．この 6 名について，他の回答を分析したところ，評価値が極端に低いということはなかった．また，自由記述を見てもタイピング速度が低下したことに対して記述されてなかった．このことから，システムに対して不満を持っているとは考えられない．この点については，自身の絶対評価ではなく，他者との相対評価（自身の成長より他者の成長の方が早く，相対的に自身が遅くなった）が影響している可能性もある．そこで，今後深くインタビューをすることで明らかにしていくつもりである．

表 3 の Q1 および Q2 の結果より，まずプログラミング経験あり群の方が，タイピング経験なし群よりもタイピングが得意であることが分かる．また，Q3，Q7 については，プログラミング経験の有無によって評価が変わることはないと言える．しかし，プログラミング経験あり群では，Q4 の実行時の意識が経験なし群より高く, Q8 の継続利用については経験なし群より低くなっていた．プログラミング経験あり群は，プログラミングでは動作確認をひとつひとつすることが大事だと身をもって知っているため Q4 の評価が高くなったと考えられる．Q8 では，自由記述に「描画する関数から書き始めたい」や「開きカッコを書いた後に閉じカッコをすぐに書きたい」,「スペースの記法が違う」など既に自身のタイピングスタイルを確立していたり，癖があったりすることにより，システムでの写経により矯正されることが苦痛と感じているからだと言える．なお，本研究はプログラミング初学者を対象としているため，このような，既にプログラミングを行ったことがある受講生は，本来は対象ではないため，システムの本質的な問題ではなく，講義の運用上の問題であると言える．

一方,「タイピングが速いとプログラムができる気になってしまった」という記述がネガティブな意見でもあったのは，想定外だった．これは，プログラミングが苦手なのにも関わらず，タイピングが高速にでき，またプログラムが組みあがったことによって，出来ていると勘違いさせてしまったことが原因で考えられる．この点については，講義中に説明を行うことで，導くことができると期待される．また，下位の受講生までの全員のランキングを掲示していたため,「順位が上がらずやる気を無くす」や「萎えた」などの意見もあった．そのため，今後はスコアの計算の仕方や，他者に共有されるランキングを制限するなど工夫を加えることで，問題を解決していく予定である．

7. オンライン講義での運用に関する議論

7.1 オンライン講義での運用結果

本システムは，プログラミングの演習講義を広く支援するものである．しかし，もともとは COVID-19（新型コロナウィルス）の拡大によって，講義の実施形態が対面か遠隔かが不明確であった第 2 著者が担当している本講義（プログラミング演習 1）において，オンライン化におけるデメリットをできるだけなくすことを考慮し，受講生の底上げを目指して生まれたものである．

一般的な対面型の講義の場合，受講生が課題に悩んでいると，隣に行ってプログラムを覗き込みながら支援することが容易である．ここで，初学者は命令のタイプミスや，カッコやカンマが間違っているなど，初歩的なケアレスミスをしがちで，受講生が質問のため手をあげたときに，その場に行ってミスを指摘することは難しくない．しかし，オンライン講義の場合，教員や TA がスムーズに質問対応することは容易ではなく，順番待ちなどを考慮すると質問を十分に捌くことができなくなる．こうした問題を踏まえて，typing.run で初歩的な命令に慣れ，記号の用法に慣れてもらっていた．その結果もあって，本講義は対面授業ではなく遠隔授業で実施されたが，講義中の課題を解いてもらう際に，初歩的なタイプミスによる問題は例年に比べかなり少なかった．

図 9 は, typing.run が 1 日でどのような時間帯に利用されたのかをグラフ化したものであるが，午前 4 時～7 時以外は広く利用されており，受講生らは暇を見つけては取り組んでおり，取り組みやすいものであったことがわかる．

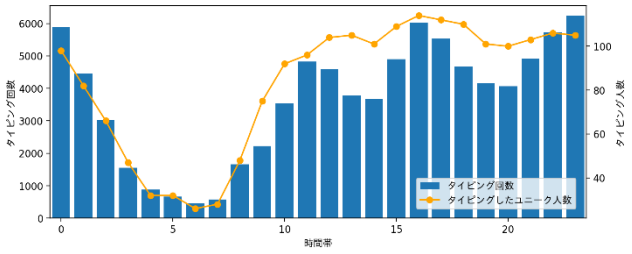

図 9　タイピングを行った時間帯に関する分析

一方，例年とは実施形態が異なるため単純に比較するこ

とは難しいが，例年と同レベルの課題について，その達成度は高かったことから，その取り組んだ効果が出ていると考えられる．

また，講義終了後にプログラミング演習 1 に関するアンケートを実施したが，93 件の講義に関するフィードバックのうち，ほぼすべてが高評価であった．また，29 件のアンケートに typing.run により助けられたことに関する記述があり，特に typing.run で楽しんだ，タイピング速度が速くなった，プログラミングが楽しくなった，他のプログラミングの講義でも是非とも使いたい，他人と競うのが楽しかったなどの言及があった．

以上のことより，本システムは，広く利用を想定しているものであるが，特にオンライン講義においてその効果を発揮するものであると考えられる．今後，対面型講義が復活した際にも運用することで，その違いについて明らかにしていく予定である．

7.2 オンラインでの運用に関する対応

本来なら運用前に，受講生を対面で集めて使い方などのチュートリアルを行うのが一般的である．しかし，オンライン環境ではできないため，システム上で大事なことは赤色の文字にしたり，フォントを大きくしたり，広く使われている共通のマークを選んだりと，わかりやすいような UI の設計にこだわった．また，問い合わせが気軽に出来るチャットも設けたうえで，TA によるログイン方法のサポートや，バグの報告やその修正，受講生からの要望を取り入れるなどの双方向なやり取りも多く行った．我々が想定していない問題が発生した時には，受講生に動画や写真，文章などで細かく情報を伝えてもらった．

本研究では，システムの公開を授業開始 1 か月半以上前に行い，またシステムの動作テストを繰り返しつつ改良を行い，受講生自体にも使用できるか確かめてもらっていたこともあり，大きな障害や問題が発生することなくスムーズに運用ができたと言える．

また，運用中は我々の監督下にないため，ツールを使った不正や JavaScript を悪用した不正な行為が起こることが公開前から予想されていた．実際に，1 名から 1000CPM 以上の人間には不可能なスコアが 5 回検出された．この受講生のアカウントを一時凍結し，不正行為を行わないように指導したうえで，凍結を解除した．一方，3 時間で 398 回，つまり，約 30 秒間に 1 回タイピングに取り組んだ受講生がいた．不自然なため不正ツールの利用を疑いアカウントを一時凍結したが，実際には本人が入力していた（この受講生は 1 つの問題に対して 1283 回取り組んでいた）．このように我々の想像以上に真剣に取り組んでくれることもあるため，データだけから不正行為かどうかを見極めるのは困難であった．

オンライン環境では，本人の実際に入力している様子が見えないうえ，直接の顔を合わせた対応ができないため，状況を把握することが容易ではない．そのため，導入に十分な時間をかけることやサポートできる環境を構築するとともに，文字入力の得手不得手などを考慮した不正判定手法が重要であると考えられる．

8. おわりに

本研究では，大学の学部 1 年生を対象としたプログラミング教育において，プログラミング特有のタイピング練習と基本命令や関数の理解と定着を行うことでプログラミング学習を支援する，プログラムタイピングシステム typing.run を提案し，運用した．その結果，タイピング速度の受講生間のばらつきが小さくなっていることでタイピングが苦手なひとの底上げが示唆されており，期間の後半ではタイピングのスコアが前半よりも成長していることが確認できた．また，アンケートからはプログラムの理解や定着に良好なことが確認でき，8 割近くの受講生が今後も使いたいと回答していた．

今後は，対応言語を広げることで，より多くの問題を載せ，幅広い分野のプログラミング学習を始めようとしている初学者に使ってもらい，タイピング速度に与える影響や学習効果を調査していく予定である．

謝辞 本システムを講義でしっかり活用してくれた明治大学総合数理学部先端メディアサイエンス学科の 8 期生および，対応してくださった TA のみなさまに感謝します．

参考文献

[1] “Processing.org”. https://processing.org/, (参照 2020-08-07).
[2] Thomas, R. C., Karahasanovic, A. and Kennedy, G. E.. An Investigation into Keystroke Latency Metrics as an Indicator of Programming Performance. 7th Australasian conference on Computing education (ACE '05). 2005, vol. 42, p. 127-134.
[3] Leinonen, J., Longi, K., Klami, A. and Vihavainen, A.. Automatic Inference of Programming Performance and Experience from Typing Patterns. 47th ACM Technical Symposium on Computing Science Education (SIGCSE ’16). 2016, p. 132-137.
[4] 中田豊久．プログラミング学習の理解度とソースコードタイピングに関する考察．研究報告コンピュータと教育（CE），2013, vol. 2013-CE-121, no. 7, p.1-8.
[5] 喜多一，岡本雅子．写経型プログラミング学習と反転授業．第 60 回システム制御情報学会研究発表講演論文集, 2016.
[6] 喜多一，岡本雅子，藤岡健史，吉川直人．写経型学習による C 言語プログラミングワークブック．共立出版, 2012.
[7] 吉長裕司，川畑洋昭．情報教育におけるキーボードリテラシーの一考察．情報処理学会論文誌，2001, vol. 42, no. 9, p. 2359-2367.
[8] Matayoshi, Y. and Nakamura, S.. Abstract Thinking Description System for Programming Education Facilitation. 22nd International Conference on Human-Computer Interaction (HCII 2020). 2020, vol. 12206, p. 76-92.
[9] “GitHub - processing-js/processing-js: A port of the Processing visualization language to JavaScript.”. https://github.com/processing-js/processing-js, (参照 2020-08-07).