Bulletin of Aichi Univ. of Education, 69 (Educational Sciences), pp. 127-131, March, 2020

EXCEL演習におけるプログラミング的思考について

松永 豊

情報教育講座

About Computational Thinking in Excel Practice

Yutaka MATSUNAGA

Department of Information Sciences, Aichi University of Education, Kariya 448-8542, Japan

1. はじめに

学習指導要領が10年ぶりに改訂され、2020年度より小学校・中学校・高校の順に実施される。様々な変更が行われるが、その中でもとりわけ特徴的にものとしては、小学校における「プログラミング教育」の導入が挙げられるだろう。

背景としては、グローバル化や人工知能・ロボットなどをはじめとした急速な技術改革など、激しい社会の変化が大きな要因である。様々な仕事が自動化されることで、今後、10年から20年程度で半分以上の職業が無くなる可能性も指摘されており、第4次産業革命とも言われている。すなわち、今の小学生には、本人が社会人になるころには無くなってしまう職業があることも教えなければならない。また、既存の職業が無くなる一方、AI技術者などが大量に足りなくなることが予測されているため、プログラミング教育の低年齢化が世界中で叫ばれているのである[1)]。

ただし、日本における小学校プログラミング教育においてはICT専用の科目が新たに作られるわけではなく、様々な科目において教科横断的に教育することが求められており、教育目標もプログラミング的思考力の育成となっている。そこで、本論文ではプログラミング的思考について再確認したのち、EXCEL演習におけるプログラミング的思考について考察する。

2. プログラミング的思考

2020年度より小学校プログラミングが始まることは決定しているが、小学校でのプログラミング教育においては、コーディングの習得を目的としていない。つまり、いわゆるプログラマの養成を目指しているわけではなく、プログラミング的思考の習得が目的となっている。

そもそも、プログラミング的思考とはいったい何のことだろう。プログラムを作成するにあたって極めて重要なファクタではあるが、無論、プログラミング能力とイコールではない。文部科学省のプログラミング教育の手引きでは次のように定義されている。

『「プログラミング的思考」は、「自分が意図する一連の活動を実現するために、どのような動きの組合せが必要であり、一つ一つの動きに対応した記号を、どのように組み合わせたらいいのか、記号の組合せをどのように改善していけば、より意図した活動に近づくのか、といったことを論理的に考えていく力」と説明されています。』

この説明文自体は、プログラミングが得意な人がプログラミングのことを意識しながら（頭の中で想像しながら）、一般人向けとして専門用語が使えない（敢えて使わない）という苦労が偲ばれる、かなりわかりにくい文章ではあるが、要は「論理的思考」と「システム思考」のことであり、より細かく言えば「分解」「順序だて」「抽象化」「一般化」「デバッグ」「評価」のことを指していると考えられる[2) 3)]。

これらの能力はプログラミングにおいて重要なことは言うまでもないが、プログラミング以外の一般的な場面でも重要な能力である。具体的な事例としては「料理」を題材に解説されることも多い。

例えば、カレーについて考えてみる。材料として、野菜や、肉、カレールーなどが必要になる。

材料	ジャガイモ、ニンジン、タマネギ、豚肉、カレールー
手順	・野菜の皮をむく ・野菜を一口大にカットする ・豚肉を一口大にカットする ・鍋に油をひく ・中火で肉を炒める ・野菜を加え、タマネギが透き通るくらいまで炒める ・水を加え 15 ～ 20 分煮込む ・火を止める ・カレールーを加える ・10 分煮込む ・完成

表１　カレー作成の手順

カレーに限らずだが､完成品の料理を食べて「これ､おいしいな｡自分で作ってみようかな｡」と考えたとき､どのような材料が必要か､どのような手順で作る必要あるか､など分析能力や論理的思考力や設計・計画能力が必要となる。細かい手順まで目を配ると

・材料（一般的には完成形と見た目がかなり異なる）の準備が必要
・肉を炒める前に油をひく必要がある
・ジャガイモの皮をむく、ニンジンの皮をむく、タマネギの皮をむく、をまとめて野菜の皮をむくと表現することが可能
・タマネギが透き通るくらいまで炒める、とは、もしタマネギが透き通るくらいでなければ炒め続け、そうでなければ炒めるのをやめる、と同等
・食べてみて好みの硬さではなかった場合は、次回、煮込みの時間を調整するとよいかも

など､ごくありふれた日常の中にも「分解」「順序だて」「抽象化」「一般化」「デバッグ」「評価」などが潜んでいることがわかる。

このように､「プログラミング的思考力」はプログラマだけに求められるものではなく、様々な問題解決において必要な能力である。そのため、小学校プログラミング教育においては、特定の科目によらず教科横断的に学ぶことが求められている。

3. EXCEL とプログラミング的思考

次にEXCELにおけるプログラミング的思考について考えてみる。周知の通り､EXCELではVBA（Visual Basic for Applications）が使用できる。VBAは最も利用されているプログラミング言語の一つであるBasic系列なので、VBAを本格的に学習する場合、当然のことながらプログラミング言語学習となるため、プログラミング的思考力のトレーニングになることは言うまでもない。しかしながら、敢えてVBAを使わないという条件でEXCEL学習をしたとしても、プログラミング的思考力は極めて重要な能力になる。VBAを用いずにセル間で四則演算や組み込み関数のみで処理する形態を敢えてプログラミングとは分ける意味で､本論文では「セル関数処理」と呼ぶことにする。

セル関数処理においても、プログラミング的センスは極めて重要となる。例えば、EXCELにおいては有益な関数が非常に多く実装されているが、それでもあらゆる場面の関数が準備されているわけではない。実際には複数の関数を組み合わせて使うことが多々ある。このとき、複数のセルを経由して計算結果を出すか、一つのセルで関数をネストさせて使うかなど、解決する方法が複数存在することが多い[4)]。

また、プログラマにとってはやや当たり前のことではあるのだが、解決の手順（アルゴリズム）が一通りとは限らないため、同じ結果を生むとしても違った材料（素材、計算式等）を組み合わせて使うことがよくある。例えば、小数点第1位で四捨五入して整数にする場合、専用関数のROUND関数を使うこともできるが、INT関数で代用することも可能である。

四捨五入用の関数を使用	=ROUND（A1,0）
専用関数以外を使用	=INT（A1+0.5）

表２　四捨五入の計算式

あらゆる桁における四捨五入が欲しい場合はROUND関数の利用が便利ではあるが、この例のように、小数点第1位を四捨五入して整数にするという場合に限れば整数化関数でうまくいくため、INT関数を使って求める人も多いだろう｡同様な事例は多々あり、このような例はVBAの使用不使用にかかわらずEXCELにはプログラミング的要素が含まれていることの証左であろう。

しかしながら、一方でセル関数処理においては一般的なプログラミング言語学習とは大きく異なる部分もある。例えば、プログラミングにおける3大制御構造について考えてみよう。

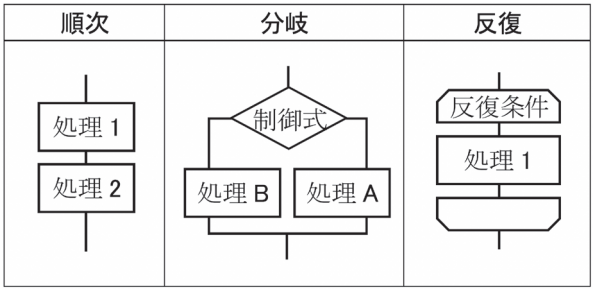

図１　プログラミング３大制御構造

3大制御構造とは「順次（逐次処理）」「分岐（条件分岐）」「反復（繰り返し）」のことで、プログラミング学習においてはアルゴリズムのイロハとして学ぶことも多い。これらの要素はセル関数処理においても重要な役割を担うが､「順次」「分岐」に関してはVBAプログラミングと特に違いはない｡「順次」に関して言えば、処理1、処理2がカスケード的に処理されていくこと、処理1と処理2の順番は重要で逆にすると違う意味になってしまうことなどはほぼ共通である。例えば、科目ごとの平均点を計算してから最大値を求める場合と、科目ごとの最大値を計算してからその平均を求める場合では結果が異なる、などは、VBAプログラミングでもセル関数処理でも違いはない｡「分岐」に関してもIF関数とIF文の違いなど多少は異なるが、動作に関してはほぼ同じである。

一方､「反復」に関しては両者で大きく異なる。そもそもセル関数処理においては本当の意味での反復は使用できない。ただし、反復の最大のメリットが手順

の効率化（同じような処理を何度も書かない）という意味と考えれば、セルのコピー（計算式のコピー）が意味合い的には近いものとなる。例えば、EXCELを用いたシミュレーションなどでは、オートフィルを用いて計算式セルをコピーすることで試行回数を増やしたことと同等の意味を持つ[4]。純粋な反復制御とは異なるが、計算式を一度書いただけで残りはコピーで済まされるわけだから、少なくとも、手順の効率化については達成できている。

4. 相対・絶対参照と面コピー

逆説的に考えれば、「セル関数処理」においては連続したセルへのコピー（オートフィルも含む）が最も重要なスキルの一つと言える。また、その際に極めて重要な概念となるのが、セルの相対参照（相対番地指定）、絶対参照（絶対番地指定）である。EXCELにおいては、セルの番地を指定する際に＄マークを付けるか付けないかが極めて重要になる。相対参照・絶対参照はセルのコピーが前提となっているため、逆に言えば、セルのコピーを使わないのであれば、＄マークの有無は一切無関係となる。つまり、正しい処理を行う計算式を組み立てることができるにも関わらず＄マークをどこにつければよいかマスターできていない人は、反復処理による効率化の恩恵をみすみす捨てていることに等しい。

セルのコピーを用いて手順を効率化するわけだから、コピー先の領域が大きければ大きいほど価値が高い。そのため、縦や横だけのコピーではなく、縦横同時のコピーが可能な問題に対して最も効果が発揮しやすい。このように縦にも横にもコピーできるコピーを本論文では面コピーと呼ぶことにする。

面コピーは、例えば、九九表がわかりやすい事例である。九九表の「いんいちがいち」の部分にのみ計算式を記述し、残りの80か所に関しては面コピーで作ることができれば大幅な効率アップが見込める。

ただし、面コピーができるような事例の場合、大抵2か所以上のセルを参照とした計算式になっており、しかも参照セルのうち縦か横のどちらか一方が固定化されるケースが多いため、初心者には最初の計算式作成の難易度が高い。今回の九九表の例でいえば、参照されるセルはA列の緑色のセルおよび1行目の黄色のセルになるため、「いんいちがいち」であるB2セルの模範解答的な計算式は　=$A2*B$1　となる。

このように最初の計算式の作成時間（主にどこに＄を付ければよいかを検討する時間）をそれなりに要する可能性はあるが、仮に＄付け作業時間が3倍かかったとしても、残り80か所は一瞬にして計算が終わるので、大幅な効率アップとなる。

	A	B	C	D	E	F	G	H	I	J
1		1	2	3	4	5	6	7	8	9
2	1	1	2	3	4	5	6	7	8	9
3	2	2	4	6	8	10	12	14	16	18
4	3	3	6	9	12	15	18	21	24	27
5	4	4	8	12	16	20	24	28	32	36
6	5	5	10	15	20	25	30	35	40	45
7	6	6	12	18	24	30	36	42	48	54
8	7	7	14	21	28	35	42	49	56	63
9	8	8	16	24	32	40	48	56	64	72
10	9	9	18	27	36	45	54	63	72	81

図２　九九表の例

面コピーのコピー元になる計算式は比較的複雑になることが多いが、よく使われる典型的な例を2例紹介する。1例目はすでに述べた九九表の形であり、このタイプを筆者は「$RC ＆ R$C型」と名付けている。もう一つは「RC ＆ R$C型」で具体例は後述の回転ずしの応用を参考にしてほしい。

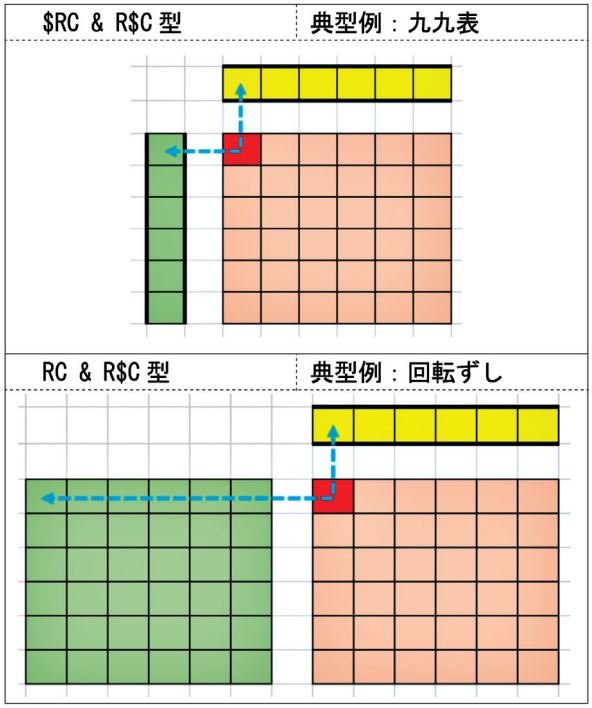

図３　面コピーでよく使われる例

$RC ＆ R$C型が、2か所の参照セルの両方とも縦か横が制限されているのに対し、RC ＆ R$C型では、2か所のうちの1か所は制限がかかっていない（＄マークが付かない）タイプである。回転ずしの例を参考にするとわかりやすい。この例では金額計算で面コピーが利用できるが、具体的にG3セルの模範解答的な計算式は　=B3*G$2　となる。

	A	B	C	D	E	F	G	H	I	J	K	L
1		枚数					金額					
2		100	150	200	300	400	100	150	200	300	400	合計
3	たろう	5	3		1		500	450	0	300	0	1250
4	はなこ		2	3			0	300	600	0	0	900
5	かずや	1	2	1	1	1	100	300	200	300	400	1300
6	さとる			3		1	0	0	600	0	400	1000
7											合計	4450

図４　回転ずしの例

5. 繰り返しが使えないもう一つの事例

連続したセルへのコピーがプログラミングの制御構造における反復に近いことは前章で説明したとおりだが、セルのコピーによる反復はループを展開しているようなものなので、膨大なセルを消費することになる。九九表のように表自体が目的の場合はよいが、途中過程を表示する必要がないケースでは無駄も多い。

VBAのようなプログラミング言語を用いる場合は、一般に変数や配列の値は表示させない、あるいは、動的に表示させることになる。セル間の計算のみで実現しようとしても、VBA（マクロも含む）の機能を使わない限りは原則として静的な値になるため、ループを展開しない形では実現できない。運よくループを使わない別の手段で実現できるかもしれないが、このとき、極めて特殊なプログラミング的思考力が必要となる場合がある。

ここで一つ、トランプのシャッフルを考えてみよう。簡単のため、カードは整数1から13の13枚と考える。典型的なシャッフルプログラム（VBA）は図5の通りである。

このシャッフル処理をVBAの使用無しに実現、すなわち「セル関数処理」だけで実現するというのが今回のミッションと考える。プログラミングが得意な人は図5の例に限らずシャッフルプログラムがすぐに頭に思い浮かんでしまうため、セル関数処理だけで作ることは意外と難しい。解答例は本論文の最後のAppendixに載せることにする。

```
Sub Shuffle（）
  Dim card（13）As Integer
  Dim i, r, t As Integer

  ' 初期化
  For i = 1 To 13
    card（i）= i
  Next

  ' シャッフル
  For i = 1 To 13
    r = Int（Rnd * 13）+ 1
    t = card（i）
    card（i）= card（r）
    card（r）= t
  Next

  ' 確認用
  For i = 1 To 13
    Cells（i, 1）= card（i）
  Next
End Sub
```

図５　シャッフルプログラム

6. まとめ

以上、本論文ではEXCELにおけるプログラミング的思考について述べた。その際、VBAを用いる場合、VBAを用いない場合について述べた。特にVBAを用いない場合は、

・セルのコピーが重要であること
・セルのコピーにおいては相対参照・絶対参照の概念が重要であること
・特に縦横へのコピー（面コピー）に耐えうる計算式の作成能力が重要であること

などについて述べた。

また、VBAを用いない場合は、純粋なプログラミング能力とは異なるプログラミング思考力が必要となることがあることを述べた。プログラミングが得意な人のほうがかえって難しく感じる演習課題も存在する。この例は、多少異なるかもしれないが、スクラッチにおける正三角形描画と近いものがあるかもしれない[5)]。スクラッチで正三角形を書く場合、ネコの動作として、100歩前進して120度右に回転、を3回繰り返せばよいが、120度を60度に間違える人が子供より大人のほうが多いらしい。これは、スクラッチでは手順上外角になるのだが、「正3角形＝内角60度」が頭に染み込んでいるため、つい間違えてしまうのだろう。子供は素直に120度を答えるケースが多いため、プログラミング的思考力という観点からは、その場で分析を行う子供のほうに軍配が上がる場合もあるということである。

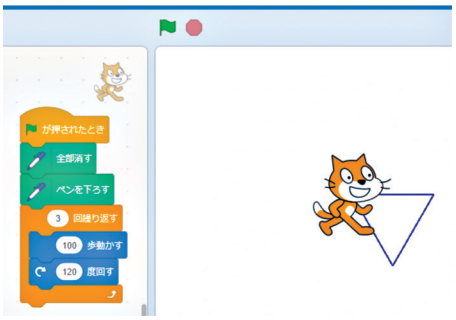

図６　スクラッチによる正三角形の描画

筆者は以前の論文でEXCELを用いることでVBA抜きでもプログラミング能力向上ができると書いたことがあるが、やや違和感のある部分も残っていた。今にして思えば、プログラミング的思考力向上と書けばしっくり来たのかもしれないと感じている。

今後、プログラミング教育の低年齢化でますますプログラミング的思考が重要になってくると思われるが、何かの参考になれば幸いである。

参考文献

1）小学校プログラミングの指導法に関する一考察，松永豊，愛知教育大学研究報告，教育科学編，66, p.157-161，2017

2）システム思考入門，飯尾 要，日本評論社，1986

3）http://shien.ysn21.jp/teacher/shien/programing_tekisikou.html

4）Excelを用いたシミュレーション演習授業，松永豊，愛知教育大学研究報告，教育科学編，60, p.187-190, 2011

5）https://coeteco.jp/articles/10512

Appendix

セル関数処理だけでシャッフルを作る場合、まず乱数を発生させ、RANK関数で順位を付ける。このとき、同一順位ができてしまうとうまくいかないので工夫が必要である。ここでは2つの方法を紹介する。

【シャッフル例　その1】

・1行目には1から13の数字が入っている

・A2=INT（RAND（）*1000）+A1/100

・A3=RANK（A2,$A$2:$M$2）

	A	B	C	D	E	F	G	H	I	J	K	L	M
1	1	2	3	4	5	6	7	8	9	10	11	12	13
2	812.01	452.02	385.03	118.04	59.05	333.06	221.07	28.08	333.09	743.1	173.11	211.12	56.13
3	1	3	4	10	11	6	7	13	5	2	9	8	12

図7　シャッフル例　その1

＜解説＞

同一順位にならないように、乱数値に影響がない範囲で同一にならない微少値を加算している。図7では、F列とI列で同じ乱数値333が発生しているが、それぞれ0.06と0.09が加算されているので同一順位にならない。

【シャッフル例　その2】

・A5=INT（RAND（）*100）

・A6=RANK（A5,$A5:$M5）+COUNTIF（A5:$M5,A5）-1

	A	B	C	D	E	F	G	H	I	J	K	L	M
4													
5	19	44	29	25	29	25	22	48	45	47	31	27	17
6	12	4	7	10	6	9	11	1	3	2	5	8	13

図8　シャッフル例　その2

＜解説＞

同一順位にならないように、同じ数字をカウントして順位に下げる工夫をしている。COUNTIFの第1引数の範囲の作り方がミソ。

（2019年9月24日受理）