# プログラミング教育の目的とあり方 (ICT教育WG資料)

久野　靖*

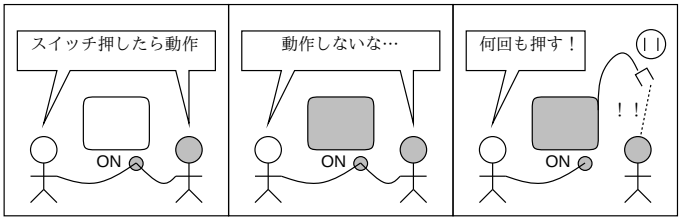
2015.2.24

## 1　プログラミングはなぜ必要?

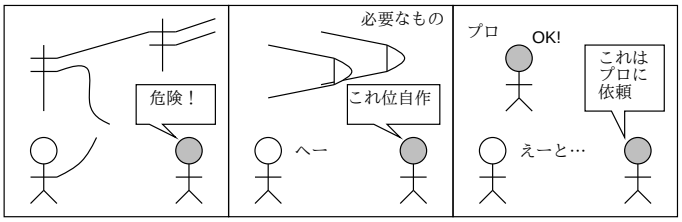
□ 類似した質問:「理科はなぜ必要?」

- べつに全員が科学者になるわけじゃないよね…

□ 仕組みを知っておくことは重要!




□ 仕組みを知っておくことは重要!

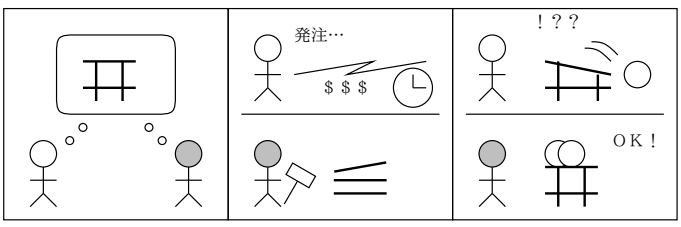



□ 今日の日本の問題…

- プロの価値が分からない → プロが育たない社会
- プロの価値が分かる社会に変えないと、未来もないですよね?

□ いま、世界が目指していること…

- 「自分の欲しいものはある程度自作できる」




□ いちばん大切なこと…「楽しい!」

- 「遊び」が人を最も成長させる!
- モノを作ることは楽しい!
- 作ったものが動いて役に立つ!
- しかもお金が掛からないし危険でもない!

□ ×「いい大学に入るためだけに何も考えずにこれをやれ」

- …大学だけ入っても役に立ちません

□ ◎「楽しいことを通じて身体も頭も鍛える」

- …新しい状況に遭遇したときに楽しめる人こそ必要
- ←プログラミングでは無尽蔵に新しいことが楽しめる
- ←コンピュータは何でもできる＋コンピュータでできることは全てプログラミングによってできる

## 2　プログラミング教育で何を学んでもらうか?

□ ×　言語の構文や機能を逐一網羅的に学んでもらう

- 古くからあるやり方だが、学習者が挫折しやすく楽しみにくい

□ △　とにかく用意したフレームワークの内容を動かさせて説明

- 「ゲームを作ろう!」「アプリを作ろう!」的
- やった気になるが「帰ってからさらに新たなものを作る」が困難

□ ◎　プログラミングするとはどういうことか分かってもらう

□ ◎　どうすればプログラミングできるのか分かってもらう

*筑波大学ビジネスサイエンス系

□ 具体的には…

- 決まった書き方の規則に従って書く → 決まった規則に従って実行される
- ごく少数の機能から成る → それらを組み合わせて何でも書ける
- 「計算式」「代入」「メソッド呼び出し」
- 「制御構造」… if、while、for

□ 重要なこと:「唯一の正解というものはない!」

- 同じ動作を何通りもの書き方で書け、どれも「正しい」
- 例:「2 つの数のうち大きい方を返す」

```
def max2(a, b)
  if a > b then return a else return b end
end

def max2(a, b)
  result = a
  if b > result then result = b end
  return result
end
```

- どの書き方で書くかは「自分で考えて決める」(美しいか?)
- (今の教育では「唯一の正しい答え」がありすぎ!!)

□ 「何が美しいか」は状況によってまったく違ったりする

- 例:「3 つの数のうち大きいものを返す」だと?

```
def max3(a, b, c)
  if a > b then
    if a > c then return a else return c end
  else
    if b > c then return b else return c end
  end
end

def max3(a, b, c)
  result = a
  if b > result then result = b end
  if c > result then result = c end
  return result
end
```

□ 「既存の部品を組み合わせる」「構造を考える」ことの重要性

```
def max3(a, b, c)
  return max2(a, max2(b, c))
end
```

□ 「唯一の正解は無い」ので「自分で考えていいと思うように書く」

- ここで難しすぎないような課題をやってもらって達成 → 「離陸」
- 例: 単純な課題だがコードにすると考えるもの (最大値、FizzBuzz 等)
- 例: 「絵を生成する」のように「結果や自分なりの正解」がすぐ分かるもの

□ ◎ プログラミングするとはどういうことか分かってもらう

□ ◎ どうすればプログラミングできるのか分かってもらう

- 「決まった書き方 (少数)」→「組み合わせると何でも」

□ 書けるようになると絶対に「楽しい」ので、そこまでの敷居を低くする

- ただし「書ける」という要件を歪めないように

## 3 ドリトル: 教育用オブジェクト指向言語

□ 「全員むけ」プログラミング教育は適した言語・処理系を使うべき

□ ドリトル: 2000 年ころ兼宗 (現・大阪電通大) と久野で開発開始

- 日本語の文字/語彙による記述 (日本語そのものではない)
- 日本語の語順に近い記述
- タートルグラフィクス (← LOGO) + オブジェクト指向
- 初心者がとまどわないような様々な配慮

□ 配布サイト: http://dolittle.eplang.jp

□ 再掲: 目的は…「プログラミングするとはどういうことか」「どうすればプログラミングできるか」分かってもらう

- 決まった規則の書き方 → 決まった規則の実行
- 少数の機能 → 組み合わせて何でも

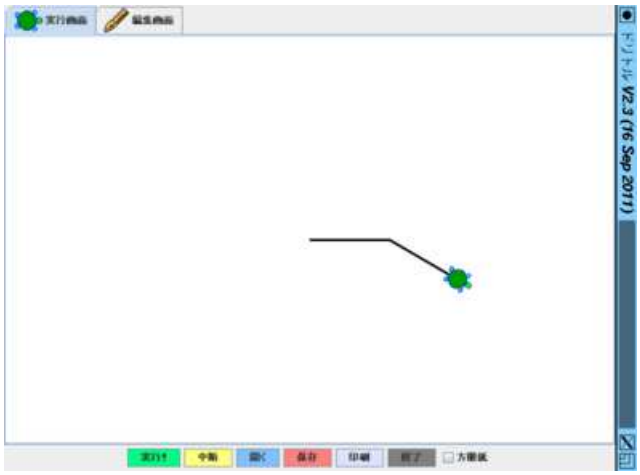
□ ドリトルにあてはめると…

- 少数の機能・書き方 → 「代入」「オブジェクトのメソッド呼び出し」

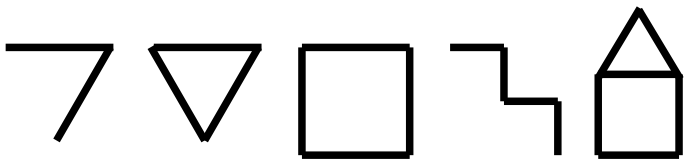
[変数 = ] obj! 引数… 命令 引数… 命令.

☐ Lesson 1.　タートルで線画を描く

かめた = タートル！作る。
かめた！100 歩く。
かめた！30 右回り 100 歩く。



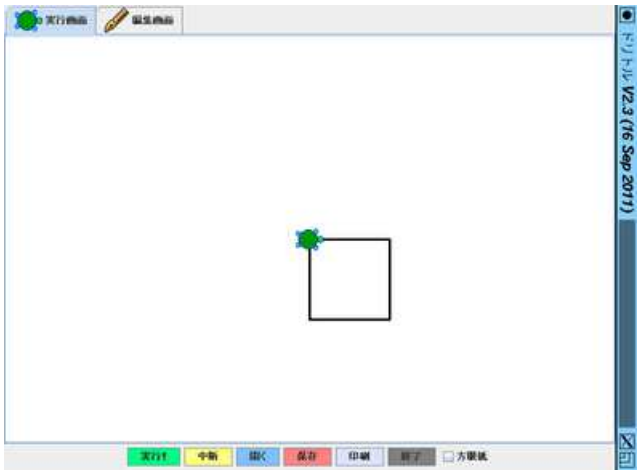
- わかったら、次のような軌跡を描いてみよう



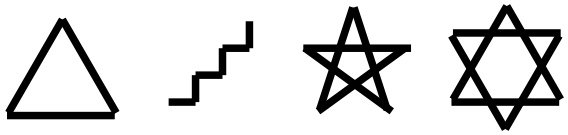
☐ タートルグラフィクスは、自分の書いたコードとその結果の対応が把握しやすいため、LOGO 以来教育に多く使われている

☐ Lesson 2.　繰り返し

かめた = タートル！作る。
「かめた！100 歩く 90 右回り。」！4 繰り返す。



- わかったら、次のような軌跡を描いてみよう



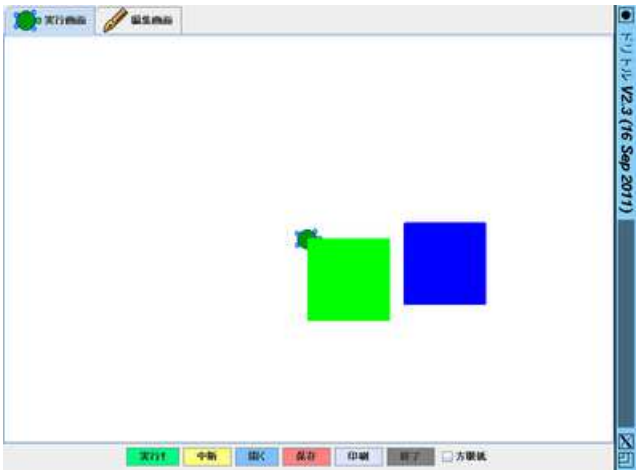
- (線を引くのを ON/OFF するには「かめた！ペンなし。」「かめた！ペンあり。」を使う)

☐ 繰り返しはプログラミングにおける強力な機能→早く教えてあげたい

- 「指定回数繰り返し」は一番分かりやすい (が、多くの言語では…)

☐ Lesson 3.　図形オブジェクト

かめた = タートル！作る。
「かめた！100 歩く 90 右回り。」！4 繰り返す。
しかく 1 = かめた！図形を作る (緑) 塗る。
しかく 2 = しかく 1！作る (青) 塗る 120 20 移動する。



- わかったら、次のようなパターンを作成してみよう

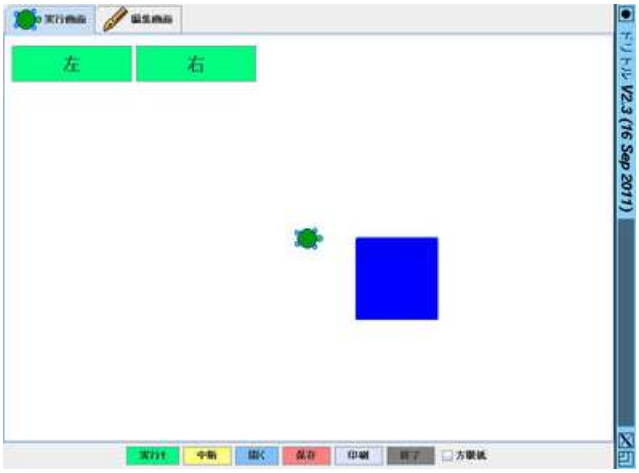
☐ オブジェクト指向的な考え方

- 「いちど作った図形は部品として使える」
- 「ものの種類ごとに使える命令は違っている」

☐ Lesson 4.　GUI 部品

かめた = タートル！作る。
「かめた！100 歩く 90 右回り。」！4 繰り返す。
しかく = かめた！図形を作る (青) 塗る。

```
ボタン 1 = ボタン！"左" 作る。
ボタン 1:動作 = 「しかく！-10 0 移動する。」。
ボタン 2 = ボタン！"右" 作る。
ボタン 2:動作 = 「しかく！10 0 移動する。」。
```




- :動作 → ボタンが押されたときの動作を指定

□ わかったら、次のようなプログラムを作ってみよう。

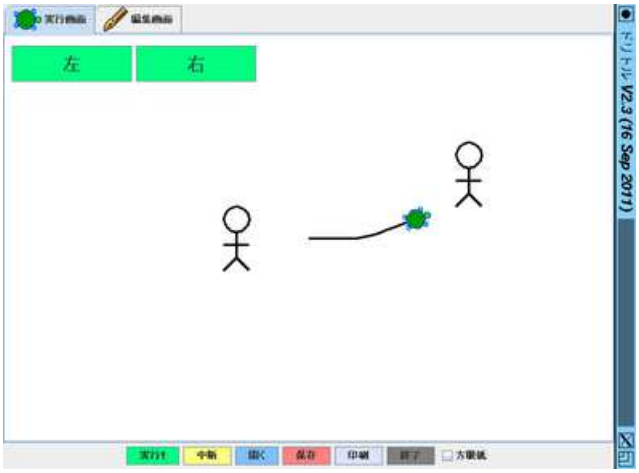
- 上の例に加えて四角を回転させるボタンがある。
- 上の例に加えて四角を「赤くする」「緑にする」ボタンがある。
- 上の例に加えて四角を「今いる位置にコピーする」ボタンがある。
- かめたを「前進」「10 度右」「10 度左」に動かすボタンがある。

□ ドリトルでは、オブジェクト指向を前面に出すことで今日的なプログラムを低い敷居で体験できるようにしている

- (まだ「数の足し算」も「x = x + 1」も使っていない!)

□ Lesson 5. タイマーとゲーム (?)

```
かめた = タートル！作る。
ボタン 1 = ボタン！"左" 作る。
ボタン 1:動作 = 「かめた！10 左回り。」。
ボタン 2 = ボタン！"右" 作る。
ボタン 2:動作 = 「かめた！10 右回り。」。
タイマー！作る「かめた！10 歩く。」実行。
かめた:衝突 = 「|x| x！消える。」。
タートル!作る ペンなし -90 00 位置 "kuno.png" 変身する。
タートル!作る ペンなし 200 80 位置 "kuno.png" 変身する。
```




- タイマー → 一定時間間隔ごとに動作を実行
- :衝突 → もの同志が衝突したときの動作 (相手を受け取れる)

□ わかったら、次のようなプログラムを作ってみよう。

- 衝突したらはね返るようにしてみる
- ブロック崩しを作る

□ ここまで来れば自分のアイデアで色々できる

□ そのほかのドリトルの特徴

- 日本語ぽい見え方 (読める)、ただし文法は明確に規定
- 文字は全角/半角を区別しない ← 構文エラーの最大の原因
- 音を出す、外部機器の制御

□ 使用経験:

- 小学校～大学までで多くの実践例
- 「プログラムを自力で書く」(←離陸) の実現に効果

## 4 教育に適したプログラミング環境

□ 多くの「実用」言語による実践が現に行われているが…

- 実用言語 (JavaScript、Java、PHP、Ruby、Python、…)
- 実際に役立つプログラムが作れるということはいいこと
- 一方で、構文や環境にまつわる「約束」が多い → 初心者にはつらい

□ プログラミング初心者が陥るつまづきのタイプ (岡本ほか)

- 「作業の自立性」に係わるつまずき → 動かす手順やエラーの対処方法などが分からないことによるつまずき
- 「具体例の認知の困難さ」によるつまずき → プログラムの動作と行いたいことの関連が認識できないことによるつまずき
- 「学習課題自体の認知の困難さ」によるつまずき → 取り上げる題材そのものが学習者にとって難しいことによるつまずき

□ これらのつまずきを少なくするような「プログラミング環境」が望まれる

- 1 番目 → 動かす手順が簡単/エラーの対処が分かりやすい (実行環境の工夫)
- 2 番目 → 行いたいことやその提示が明確 (例: タートルグラフィクス)
- 3 番目 → 取り上げる題材が難しくなくて済む (例: 固定数反復)

□ いくつかの代表的な環境

- アルゴロジック (JEITA) → ブロック型の手順で「旗を取る」パズル
- ビスケット (NTT 研究所) → 絵のルールによる変形
- Scratch(MIT) → ブロック構造により構文誤りなしにコードを組み立て
- ブロック型は他にも「プログラミン」「MoonBlock」など多数
- ドリトル (前述) → テキスト型、オブジェクト指向

□ どれが唯一ということはなくそれぞれに特徴

- 先に述べた「つまずき」ができるだけ軽減されるような配慮
- 最終的な目標は「離陸」

□ なぜ「離陸」をめざすか…(復習)

- コードを組めること＝＝自立したコンピュータ使用者になること
- 「楽しいことを通じて身体も頭も鍛える」

## 5 まとめ

□ 本日のあらましは…

- プログラミングは必要!
- プログラミング教育で目標とすべきことは? 進め方は?
- ドリトル: 教育むけプログラミング環境の例
- 教育向けプログラミング環境に求められること

□ すべての子供に「離陸」体験をぜひ!

- コードを組めること＝＝自立したコンピュータ使用者になること
- 「楽しいことを通じて身体も頭も鍛える」

## 参考文献


[1] 阿部和弘, 小学生からはじめるわくわくプログラミング, 日経 BP, 2014.

[2] 阿部和弘, 橋爪香織, 谷内正裕, 5 才からはじめるすくすくプログラミング, 日経 BP, 2014.

[3] アルゴロジック, http://home.jeita.or.jp/is/highschool/algo/

[4] 兼宗 進, 久野 靖, ドリトルで学ぶプログラミング— グラフィックス, 音楽, ネットワーク, ロボット制御 — 第 2 版, イーテキスト研究所, 2011.

[5] 岡本雅子, 喜多 一, プログラミングの「写経型学習」における初学者のつまずきの類型化とその考察, 滋賀大学教育実践研究指導センター紀要, vol. 22, pp. 49-53, 2014.

[6] プログラミング言語ドリトル, http://dolittle.eplang.jp

[7] ビスケット, http://www.viscuit.com/