

コンピュータ プログラミング

Python でアルゴリズムを
実装しながら問題解決を行う

電子情報通信学会◎編

富樫 敦 著

コロナ社

▶電子情報通信学会 教科書委員会 企画委員会◀

- 委員長 ————— 原 島 博 (東京大学名誉教授)
- 幹事 ————— 石 塚 満 (東京大学名誉教授)
(五十音順)
- 大石進一 (早稲田大学教授)
- 中川正雄 (慶應義塾大学名誉教授)
- 古屋一仁 (東京工業大学名誉教授)

▶電子情報通信学会 教科書委員会◀

- 委員長 ————— 辻 井 重 男 (東京工業大学名誉教授)
- 副委員長 ————— 神 谷 武 志 (東京大学名誉教授)
- 宮 原 秀 夫 (大阪大学名誉教授)
- 幹事長兼企画委員長 ——— 原 島 博 (東京大学名誉教授)
- 幹事 ————— 石 塚 満 (東京大学名誉教授)
(五十音順)
- 大石進一 (早稲田大学教授)
- 中川正雄 (慶應義塾大学名誉教授)
- 古屋一仁 (東京工業大学名誉教授)
- 委員 ————— 122名

(2017年12月現在)

刊行のことば

新世紀の開幕を控えた 1990 年代、本学会が対象とする学問と技術の広がりや興行きは飛躍的に拡大し、電子情報通信技術とほぼ同義語としての“IT”が連日、新聞紙面を賑わすようになった。

いわゆる IT 革命に対する感度は人により様々であるとしても、IT が経済、行政、教育、文化、医療、福祉、環境など社会全般のインフラストラクチャとなり、グローバルなスケールで文明の構造と人々の心のありさまを変えつつあることは間違いない。

また、政府が IT と並ぶ科学技術政策の重点として掲げるナノテクノロジーやバイオテクノロジーも本学会が直接、あるいは間接に対象とするフロンティアである。例えば工学にとって、これまで教養的色彩の強かった量子力学は、今やナノテクノロジーや量子コンピュータの研究開発に不可欠な実学的手法となった。

こうした技術と人間・社会とのかかわりの深まりや学術の広がりを踏まえて、本学会は 1999 年、教科書委員会を発足させ、約 2 年間をかけて新しい教科書シリーズの構想を練り、高専、大学学部学生、及び大学院学生を主な対象として、共通、基礎、基盤、展開の諸段階からなる 60 余冊の教科書を刊行することとした。

分野の広がりに加えて、ビジュアルな説明に重点をおいて理解を深めるよう配慮したのも本シリーズの特長である。しかし、受身的な読み方だけでは、書かれた内容を活用することはできない。“分かる”とは、自分なりの論理で対象を再構築することである。研究開発の将来を担う学生諸君には是非そのような積極的な読み方をしていただきたい。

さて、IT 社会が目指す人類の普遍的価値は何かと改めて問われれば、それは、安定性とのバランスが保たれる中での自由の拡大ではないだろうか。

哲学者ヘーゲルは、“世界史とは、人間の自由の意識の進歩のことであり、… その進歩の必然性を我々は認識しなければならない”と歴史哲学講義で述べている。“自由”には利便性の向上や自己決定・選択幅の拡大など多様な意味が込められよう。電子情報通信技術による自由の拡大は、様々な矛盾や相克あるいは摩擦を引き起こすことも事実であるが、それらのマイナス面を最小化しつつ、我々はヘーゲルの時代的、地域的制約を超えて、人々の幸福感を高めるような自由の拡大を目指したいものである。

学生諸君が、そのような夢と気概をもって勉学し、将来、各自の才能を十分に発揮して活躍していただくための知的資産として本教科書シリーズが役立つことを執筆者らと共に願っている。

なお、昭和 55 年以来発刊してきた電子情報通信学会大学シリーズも、現代的価値を持ち続けているので、本シリーズとあわせ、利用していただければ幸いです。

ii 刊 行 の こ と ば

終わりに本シリーズの発刊にご協力いただいた多くの方々に深い感謝の意を表しておきたい。

2002年3月

電子情報通信学会 教科書委員会
委員長 辻 井 重 男

まえがき

本書は、「プログラミング」の教科書である。より正確に言えば、問題解決のためのアルゴリズムとその実装の教本である。プログラミングとはプログラムを作成する総合的なプロセスであるが、プログラムはコンピュータに限った術語だけではない。入学式のプログラム、TVプログラム、教育プログラムなどのプログラムもある。共通するのは、ことの順番という概念であり、どの使用例でも順序が重要となる。以上から、「コンピュータプログラムとは、コンピュータにやらせたい処理を順番に書いたもの」となる。昨今では、書かれた通りにプログラムは必ずしも実行されない。本書では、人工知能分野や教育分野で一番利用頻度が高いPythonというプログラミング言語を採用し、プログラミング教育を行う。

人工知能、深層学習の社会的インパクトの大きさから、プログラミングの重要性が全世界的に認識されつつある。プログラミングを習得すれば、誰でもAIプログラムを書けるようになったのである。日本においても、ついに2020年度にプログラミングが小学校教育で必修化になった。中学では2021年度に、高校教育では2022年度に必修化する。大学教育においても、文部科学省の指導もあり、プログラミングを全学必修とする大学も出てきた。

プログラミング言語の歴史をひも解くと、年々プログラミングは人間に寄り添ってきた。しかし、自然言語で自由奔放にその解法アルゴリズムを書いても、いまだにコンピュータは処理できない。まだ人間とコンピュータの間には、ギャップがある。その中でも、Pythonは、人間の思考法や表現法に比較的近い特徴を有するプログラミング言語である。本書では、アルゴリズムの記述・実装としてPythonを採用し、そのプログラミングスタイルに従う。

本書はプログラミングの入門書であるが、文法書ではない。問題解決という目的のために、その背後の関連知識も身に付けながら、その解法をアルゴリズムとして思索し、プログラムによって実現する。つまり、本書の目的は、プログラミングによる問題解決法を習得することを目的とする。つまり、昨今のデータサイエンティストのように、課題を解決していく能力を涵養したい。そのためには、数学、エンジニアリング、サイエンスの素養も身に付けなければならない。

有能なプログラマーは、数学者のように数学という武器を利用して頭に浮かんだアイデアを表現・推論し、ときには証明・計算もする。またときには、エンジニアのようにことがらを設計し、システムまで組み立て全体をテスト・評価する。またときには、自然科学者のように自然界を観察し、仮説を立て予測を評価・証明する。プログラマーにとって最も重要な能力は、問題解決能力である。問題の所在を確かめ、定式化し、創造的に解決法を思考し、正確かつ効率よく問題を解決していく。プログラミングは、問題解決能力を磨く絶好のタスクであ

る。

以上の思いを描いて、8章構成で本書の目的を達成する。最後の8章では、代表的な問題をプログラミングを通して解決していく。そこで重要なのが、モジュール化、抽象化の概念であり、その思想の具現化であるオブジェクト指向の概念を7章で学ぶ。プログラムは、データ構造と制御構造からなる。Pythonの柔軟で強力なデータ構造力を4章で学ぶ。制御構造については、残りの章で学ぶ。1章は序章であるが、2章から4章は、チュートリアル的存在である。つまり、4章まで読めば、一通りのプログラムが書ける能力を有するように工夫した。3章の前半では、局所変数・大域変数、スコープといった関数の重要な概念を補完する。また、後半では、エラーと例外処理に切り込む。5章で条件分岐とループ構造、6章で簡易的抽象化・モジュール化の具体化である関数を習得し、再帰関数でその章を締めくくる。学習者の利便性を考慮し、付録ではPythonのデータ構造・主な関数を中心にまとめた。必要に応じて、参照していただきたい。

本書執筆に当たり、本書査読者の教科書委員会幹事・東京大学名誉教授の石塚満先生には、著者が気づかない細部にまで目を通していただいた。また、公立大学法人宮城大学の須栗裕樹教授には、草稿段階の粗削りのテキスト並びに校正稿を精読いただき、適切なコメントを頂いた。

最後に、本書執筆の機会を与えていただいた電子情報通信学会教科書委員会の皆様、そして怠惰な執筆者を辛抱強く待ち続け、本書完成までお付き合いいただいた委員会・コロナ社の担当者には最大の感謝を表したい。また、刊行にあたりお世話になったコロナ社に敬意を表する次第である。

2022年2月

富 樫 敦

【プログラムなど、電子データのダウンロード】

修学者の利便性を考慮し、本書に掲載したプログラムと簡単な解説、補足事項、章末問題の解答などを、以下の本書の書籍詳細ページ（コロナ社のWebページから書名検索でもアクセス可能）からダウンロードできるようにした。

<https://www.coronasha.co.jp/np/isbn/9784339018226/>

すべて、Anaconda社のJupyterNotebook形式に統一し、実行しながら学習できるようにした。本書の修正や本書にない情報も掲載したので活用していただきたい。

目次

1. プログラミングとは

1.1 はじめに	2
1.2 プログラミングとは	2
1.3 アルゴリズムとプログラム	3
1.4 問題解決とアルゴリズムに関する基礎知識	5
1.4.1 探索問題	5
1.4.2 ソーティング問題	6
1.4.3 最短経路問題	6
1.4.4 最適化問題	7
1.5 計算可能関数	7
本章のまとめ	8
理解度の確認	8

2. プログラミング・チュートリアル (制御フロー編)

2.1 数とその表現	10
2.1.1 数とf文字列	10
2.1.2 print() 関数	11
2.1.3 指数形式の浮動小数点数表示	11
2.1.4 タプル・パッキング&シーケンス・アンパッキング	12
2.2 算術演算と式の計算	13
2.2.1 算術演算	13
談話室 Jupyter Notebook のセル	15
2.2.2 式の計算	15
2.2.3 フェルマーの小定理	15
2.3 変数・代入・平方根	17

談話室 コメントを追加する	17
2.4 関数定義	19
2.4.1 def文による無名関数定義	19
2.4.2 ラムダ式による関数定義	21
2.5 条件分岐	22
2.5.1 条件分岐とは	22
2.5.2 うるう年	23
2.5.3 偽造硬貨・天秤問題	24
2.6 繰り返し (ループ)	29
2.6.1 繰り返し構造のパターン	29
2.6.2 ユークリッドの互除法 (while文の適用例)	30
談話室 水汲み問題	33
2.6.3 二分法による平方根の求め方 (while文の適用例)	34
2.6.4 じゃんけんプログラム	36
本章のまとめ	39
理解度の確認	40

3. プログラミング・チュートリアル (制御フロー編・発展編)

3.1 関数 (発展的課題)	42
3.1.1 関数のグラフ	42
3.1.2 関数の図的表現	43
3.1.3 素数判定	45
3.2 エラーと例外処理	47
3.2.1 構文エラー	47
3.2.2 例外	48
3.2.3 例外処理	49
3.2.4 例外発生	51
3.2.5 assert文	51
3.3 アルゴリズムとプログラム	52
3.3.1 アルゴリズムとは	52
3.3.2 アルゴリズムの記述	53
3.3.3 アルゴリズム解析と計算量	54

3.3.4 時間計算量からみたアルゴリズムの主なクラス	54
本章のまとめ	55
理解度の確認	56

4. プログラミング・チュートリアル (データ構造編)

4.1 変数とその評価	58
4.1.1 代入文の処理系上の仕組み	58
4.1.2 変数への値の渡し方と参照渡し	60
4.2 データ型	62
4.2.1 用途と特徴に応じたデータ構造の分類	62
4.2.2 数・数値	62
4.2.3 リスト	66
4.2.4 文字列	68
4.2.5 タプル	70
4.2.6 range	72
4.2.7 辞書 (ディクショナリ)	73
4.2.8 集合	75
4.3 グラフの実現とアルゴリズム	76
4.3.1 グラフの実現法 (表現法)	77
4.3.2 到達可能性問題	79
4.3.3 グラフ問題: ダイクストラの最短経路アルゴリズム	82
4.4 オブジェクト指向	87
4.4.1 オブジェクト指向プログラミングとは	87
4.4.2 データ構造スタックとキュー	87
4.4.3 スタックのクラス定義と応用	88
4.4.4 キューのクラス定義と応用	92
本章のまとめ	95
理解度の確認	96

5. 条件分岐と繰返し

5.1 条件分岐	98
5.1.1 選択：if 文による条件分岐	98
5.1.2 偽造硬貨問題	98
5.2 条件的繰返し処理（while ループ）	102
5.2.1 while 文による条件的繰返し	102
5.2.2 ニュートン・ラフソン法	104
5.3 反復可能オブジェクトによる繰返し（for ループ）	107
5.3.1 for 文の構造	107
5.3.2 繰返し回数の指定法	108
5.3.3 繰返し処理例：数が作る美	108
5.4 繰返しからの脱却	110
5.4.1 脱却のイメージ：三つのケース	110
5.4.2 プログラム例	111
5.5 内包表記と繰返し	112
5.5.1 集合の外延的表記と内包的表記	112
5.5.2 リスト内包表記	112
本章のまとめ	114
理解度の確認	114

6. 関数と再帰

6.1 関数の基礎	116
6.1.1 関数定義	116
6.1.2 グローバル変数	118
6.2 Python での関数流儀	118
6.2.1 位置引数・キーワード引数・デフォルト値	118
6.2.3 関数の中で補助関数を利用	121
6.2.4 可変長引数	124
6.3 再帰関数	127
6.3.1 再帰関数とは	127
6.3.2 階乗関数	128
6.3.3 フィボナッチ数列	129

6.3.4 回文	134
6.4 再帰関数の効率化	140
6.4.1 再帰関数の実行の仕組み	140
6.4.2 再帰関数の効率化	141
本章のまとめ	143
理解度の確認	144

7. オブジェクト指向プログラミング

7.1 オブジェクト指向とは	146
7.2 クラスの定義	147
7.2.1 クラス定義	147
7.2.2 インスタンス属性・インスタンスメソッド	150
7.2.3 クラス属性・クラスメソッド	152
7.3 クラスの継承	153
7.4 オブジェクト指向プログラミングの醍醐味	155
7.4.1 イテラブルクラス・イテレータクラスの実装	155
7.4.2 カプセル化	158
本章のまとめ	159
理解度の確認	160

8. 問題解決とプログラミング

8.1 アルゴリズム設計戦略	162
8.1.1 問題解決とアルゴリズム設計	162
8.1.2 縮小統治法による有名人の問題解決	163
8.2 探索問題	165
8.2.1 線形探索アルゴリズム	165
8.2.2 二分探索アルゴリズム	167
8.2.3 数以外の探索問題	168
8.3 ソーティングアルゴリズム	169
8.4 貪欲アルゴリズム	169
8.4.1 コイン問題	170

8.4.2	ナップザック問題	173
8.4.3	ダイクストラの最短経路アルゴリズム	177
8.5	動的計画法	178
8.5.1	動的計画法とは	178
8.5.2	コイン問題	178
	本章のまとめ	185
	理解度の確認	185

付 録

1.	演 算	186
2.	データ型一般	186
3.	シーケンス型の操作	187
4.	データ型独自の固有操作	189
5.	組み込みデータ型間の変換	191
6.	deque	191

引用・参考文献	192
索 引	194

【問題】

外見からは区別できない8枚の硬貨がある。このうち1枚は偽造硬貨で本物とは重さが異なる。重りなしの天秤を3回使って偽造硬貨を見つけ出し、同時に本物より重いか軽いかを見極めたい。その方法とは？

【解法】 図 5.1.2 の決定木で与える。決定木に付随したラベルについては、例 2.5.6 と同様、[Levitin11] を参考にした。木の外側のラベルは偽造硬貨の可能性を表し、その解釈は例 2.5.6 と同様である。ただし、本例では偽造硬貨が重い場合もあるので、例えば $G+$ は硬貨 G は偽造硬貨であり、重い可能性があることを示す。また、 $G\pm$ は G が偽造硬貨で重い場合と軽い場合の両方の場合があることを意味する。

アルゴリズムの本質は、8枚の硬貨を3分割に近い $3:3:2$ に分けるところにある。 $4:4$ の2分割にしたら、3回では見つけられない。決定木中のラベル No Way は、問題設定上あり得ない場合を意味する。初めに、決定木が左右対称になっていることに注意されたい。簡単な $ABC = DEF$ の場合は、残りの G, H のいずれかが偽造硬貨である。正しい A と偽造硬貨の可能性がある G を比較し、秤の傾き方によって G, H のどちらが偽造硬貨でかつその軽重が判定できる。 $ABC < DEF$ の場合は、今度は G, H が本物の硬貨となる。この場合の可能性は $A - B - C - D + E + F +$ である。次に、 A と D を皿にそのまま残し、重い可能性のある EF を左の皿に移し、右の皿には正しい GH を乗せ、 $AEF : DGH$ の比較を行う。軽重の可能性を併記すると $A - E + F + : D + GH$ となる。 $A - E + F + = D + GH$ ならば軽い可能性がある残りの B, C のいずれかが偽造硬貨で本物より軽い。 $A - E + F + < D + GH$ ならば、 A が軽い偽造硬貨か、あるいは D が重い偽造硬貨かのどちらかである。続く分岐は、 $A : G$ のように正しい硬貨 G と比較すれば決着が付く。 $A - E + F + > D + GH$ ならば E, F のいずれかが偽造硬貨で本物より重い。残された処理は、 G との比較である。

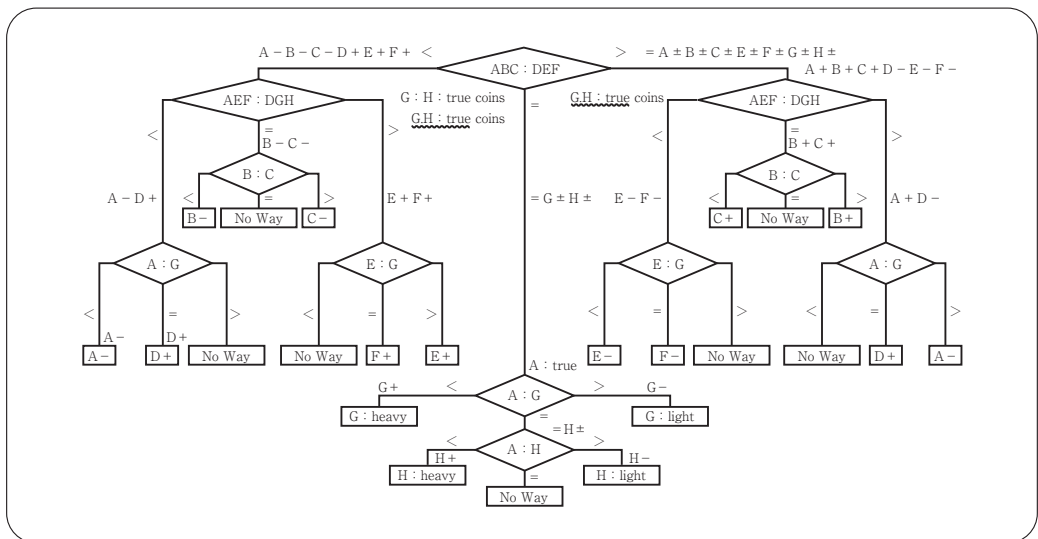


図 5.1.2 決定木

$0 \times 9 + 1 = 1$ $1 \times 9 + 2 = 11$ $12 \times 9 + 3 = 111$ $123 \times 9 + 4 = 1111$ $1234 \times 9 + 5 = 11111$ $12345 \times 9 + 6 = 111111$ $123456 \times 9 + 7 = 1111111$ $1234567 \times 9 + 8 = 11111111$ $12345678 \times 9 + 9 = 111111111$ $123456789 \times 9 + 10 = 1111111111$	$1 \times 8 + 1 = 9$ $12 \times 8 + 2 = 98$ $123 \times 8 + 3 = 987$ $1234 \times 8 + 4 = 9876$ $12345 \times 8 + 5 = 98765$ $123456 \times 8 + 6 = 987654$ $1234567 \times 8 + 7 = 9876543$ $12345678 \times 8 + 8 = 98765432$ $123456789 \times 8 + 9 = 987654321$
(1)	(2)
$9 \times 9 + 7 = 88$ $98 \times 9 + 6 = 888$ $987 \times 9 + 5 = 8888$ $9876 \times 9 + 4 = 88888$ $98765 \times 9 + 3 = 888888$ $987654 \times 9 + 2 = 8888888$ $9876543 \times 9 + 1 = 88888888$ $98765432 \times 9 + 0 = 888888888$	$1 \times 9 + 2 = 11$ $12 \times 9 + 3 = 111$ $123 \times 9 + 4 = 1111$ $1234 \times 9 + 5 = 11111$ $12345 \times 9 + 6 = 111111$ $123456 \times 9 + 7 = 1111111$ $1234567 \times 9 + 8 = 11111111$ $12345678 \times 9 + 9 = 111111111$ $123456789 \times 9 + 10 = 1111111111$
(3)	(4)

図 5.3.2 数が作る美

実行結果

```

1 x 9 + 2 = 11
12 x 9 + 3 = 111
123 x 9 + 4 = 1111
1234 x 9 + 5 = 11111
12345 x 9 + 6 = 111111
123456 x 9 + 7 = 1111111
1234567 x 9 + 8 = 11111111
12345678 x 9 + 9 = 111111111
123456789 x 9 + 10 = 1111111111

```

[コードの説明] (スクリプト 5.3.2)

1. 文字列 s は、#3 により、i 行目の 12...i を作り出している。
2. s を使って、文字列 t で左辺の式を形成している。左辺の最後の数が一桁か二桁に応じて前の空白の数が異なるので、#4-#8 の条件分岐で場合分けしている。
3. #8 の右辺の式は、11...1 (1 が i + 1 個) を生成するための巧妙な式である。i = 5 の場合は、111111 = 12345 * 9 + 6、となる。
4. #9 の print 関数では、最初に半角で「18 文字分として t を右詰め ({t: > 18})」で出力、その後「=」を出力、最後に「10 文字分として num を左詰め」({num: < 10}) で出力

2. 帰納的ケースの部分 (#4-#5)
3. `test_fib(m, n)`は `fib()`のテスト用の関数であり, `fib(m), fib(m+1), \dots, fib(n)`を求めて出力する.

(3) フィボナッチ数列の視覚化

【例 6.3.5】 (フィボナッチ数列のグラフ) フィボナッチ数列の視覚化として, 数列を棒グラフで表してみる. `matplotlib.pyplot`を使ったグラフ描画については, 例 3.1.1 である程度説明したので再度参照してほしい.

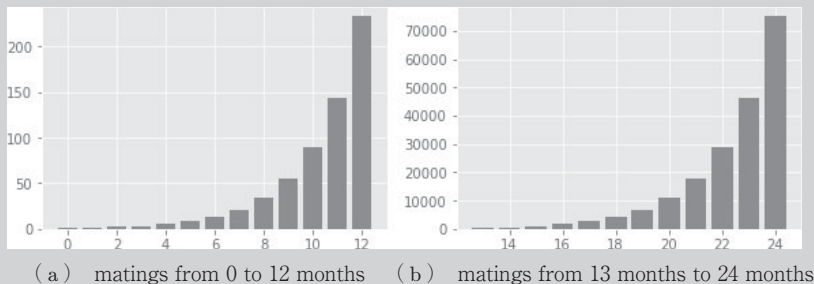
スクリプト 6.3.6

```

1 import matplotlib.pyplot as plt
2 import numpy as np
3 plt.style.use('ggplot')
4
5 x = list(range(24+1))
6 y = [fib(n) for n in range(24+1)]
7 fig = plt.figure(figsize=(10, 3))
8 ax1 = fig.add_subplot(121)
9 ax1.bar(x[:13], y[:13])
10 ax1.set_title('matings from 0 to 12 months')
11 ax2 = fig.add_subplot(122)
12 ax2.bar(x[13:], y[13:])
13 ax2.set_title('matings from 13 months to 24 months')
14 plt.show()

```

実行結果



2年後にもなると, つがいの数は何と約7万5千まで上昇する. 正確には, 75,025. さらに, 3年後, 10年後と知りたくなるが, 結果を得るまで相当待たねばならないので, 6.4節の効率化を習得してからこの難題に挑戦する.

定理 6.3.1 (フィボナッチ数列の一般項 (ビネの公式)) フィボナッチ数列の n 番目の数 f_n は, 式 (6.3.3) で表すことができる. この式は 1843 年にビネ (Jacques Philippe Marie Binet) が発表したことからビネの公式 (Binet's Formula) と呼ばれるが [Cormen09], それ以前の 1730 年 (ド・モアブル) [Cormen09], 1765 年 (オイラー) [Cormen09] にも発表されており, ビネが最初の発見者ではなさそうである. スクリプト 6.3.7 は, ビネの公式を用いてフィボナッチ数列を求める関数である.

```
pyramid(9)
```

実行結果

```

  1 x 1      =      1
 11 x 11     =     121
111 x 111    =    12321
1111 x 1111  =   1234321
11111 x 11111 =  123454321
111111 x 111111 = 12345654321
1111111 x 1111111 = 1234567654321
11111111 x 11111111 = 123456787654321
111111111 x 111111111 = 12345678987654321

```

関数 `layer()` を使って、数の回文ピラミッドを作成した。10進数なので、ピラミッドタイプの回文は9が最下層の段になる。forループの関数適用 `layer(i, n-i+1)` により、`i`が増えるに従って `11...1` の長さが長くなり、同時に '=' の前後の空白の長さが短くなっていく。

(3) どんな数でも回文になる？

[例 6.3.9] (どんな数でも回文になる?) 次の操作を通して、どんな数でも回文になるかどうかを確かめる。

[回文への変換操作]

- 最初に提示された数が回文なら操作は直ちに終了。
- その数とその数を逆順にした数を足し、回文になるかどうかを確かめる。
- 回文になればそこで終了する。
- 回文にならない場合は、2.以降の操作を続ける。

例えば、1432 の場合、数そのものは回文ではない。そこで、その数を逆順にした数 2341 を足すと

$$1432 + 2341 = 3773(0)$$

という回文になる。数の括弧の中の数字は足し算の際の繰り上げ回数を表す。それでは 382 の場合

$$382 + 283 = 665(1), 665 + 566 = 1231(3), 1231 + 1321 = 2552(0)$$

と3ステップ目で回文になった。足し算の結果繰り上げがなかった場合は、必ず回文になる(その証明は、自明)。しかし、この条件は結果が回文であるための十分条件ではあるが、必要条件ではない。例えば、 $47 + 74 = 121(2)$ のように2回の繰り上げがあったが、1ステップ目で回文になる数 47 がある。59 は

$$59 + 95 = 154(2), 154 + 451 = 605(1), 605 + 506 = 1111(2)$$

のように各ステップで繰り上げがあるが、3ステップ目で回文になった。それでは、この操作を自動化してみる。スクリプト **6.3.12** は、文字列表現の整数 `s`, `t` を整数として足した場合のその和と繰り上げ回数を返す関数 `carry()` である。

8.5 動的計画法

8.5.1 動的計画法とは

動的計画法とは、完全である全数探索を基本にし、縮小統治法と記憶化を組み合わせた手法である。

【アルゴリズム 8.5.1】(動的計画法) 動的計画法 (dynamic programming) は、問題を重複する幾つかの部分問題に再帰的に分割し、かつ計算結果を記憶化し、同じ処理の繰り返しを避ける方法である。

8.5.2 コイン問題

(1) コイン問題 [動的計画法]

動的計画法の例として、貪欲アルゴリズム (8.4 節) で学んだコイン問題を取り上げる。

【問題 8.5.1】(コイン問題) [動的計画法] コイン問題とは、コインの額面の集合 $Coins = \{c_1, c_2, \dots, c_k\}$ が設定されたもとの、支払い金額 pay をできるだけ少ない枚数のコインで支払う方法を答える問題であった (問題 8.4.1 参照)。

8.4 節では、貪欲アルゴリズムを使って問題を解決した。解ったことは、コインの種類がユーロコインの場合は、貪欲アルゴリズムは常に最適解を与えた。しかし、一般の場合は、アルゴリズムの解は、必ずしも最適解とは限らなかった。この節では、動的計画法を用いてコイン問題を効率的に解決する。貪欲アルゴリズムと違い、動的計画法の場合、硬貨の種類、支払い金額に左右されずに、常に最適解を与える。動的計画法によるアルゴリズムは、再帰的定義による総当たり法を採用するが、記憶化という方法によって同じ計算を繰り返さない。つまり、動的計画法の本質は、再帰的手法によるオーバーラップのある部分問題への分割と記憶化による効率化である。

(2) 再帰的定式化

【再帰的定式化】 動的計画法を適用する場合、問題の再帰的定式化 (recursive formulation) が必要となる。問題の解を分割したオーバーラップする部分問題の解から再構築する。コイン問題では、「支払い金額 pay を支払うのに必要なコインの最小枚数？」が問題である。そこで、関数 $c_num(pay, Coins)$ は、コインの種類を表した集合 $Coins$ を使用した場合、支払い金額 pay に必要なコインの最小枚数とする。当然、関数の値は設定したコインの種類に依存

【あ行】

値渡し60
 アリとキリギリス74
 アルゴリズム3, 52, 53
 アルゴリズム設計162
 アルゴリズム設計戦略162
 イソップ童話74
 位置引数119
 イテラブルオブジェクト107
 イテラブルクラス155
 イテレータ80
 イテレータクラス155
 イニシャライザ89, 150
 イミュータブル60, 70
 インスタンス147
 インスタンス化147
 インスタンス属性89, 150
 インスタンス変数89, 148
 インスタンスメソッド150
 うるう年23
 エラー47
 エラトステネスの篩76
 演算子14
 円周率40
 黄金比36, 132
 オブジェクト146
 オブジェクト指向87
 オブジェクト指向パラダイム146
 オブジェクト指向プログラミング87
 重み付き有向グラフ77
 親クラス153

【か行】

外延的記法112
 外延的表記112
 階乗関数128
 回文134
 帰りがけ順90
 加算13
 華氏20
 数の回文135
 数の回文ピラミッド136
 カプセル化158
 可変長引数124
 仮引数19, 116
 関数3, 19
 関数アノテーション116
 関数定義19, 116
 関数定義・スコープ116

関数ボックス44
 関数本体116
 関数名116
 関数呼び出し116
 キーワード引数14, 119
 記憶化142, 181
 基数63
 偽造硬貨・天秤問題24
 偽造硬貨問題98
 基底63
 基底クラス153
 基底ケース127
 帰納的ケース127
 逆ポーランド記法89
 キュー87, 92
 局所変数44
 クイックソート123, 169
 組込み関数120
 クラス3, 147
 クラス属性152
 クラス定義147
 クラス変数148, 152
 クラスメソッド152
 グラフ42, 77
 グラフの最短経路問題82
 繰返し3, 29
 グローバルスコープ44
 グローバル変数44, 118
 計算可能8
 計算可能関数7
 計算量54
 継承153, 158
 継続文字12
 ゲッター151
 減算13
 厳密性52
 コイン問題103, 170, 178, 184
 後行順序訪問90
 合成数45
 構造的帰納法127
 後置記法90
 構文エラー47
 子クラス153
 コメント17
 コンストラクタ89, 147, 150

【さ行】

再帰関数127
 最大公約数32
 最短経路問題6, 82
 最適化問題7
 最良選択104

最良探索アルゴリズム169
 最良法162
 サブクラス153
 三項演算子14
 算術演算3, 13
 参照58
 参照アドレステーブル58
 参照渡し60
 シーケンス71
 シーケンス・アンパッキング12, 71
 時間計算量54
 辞書73
 辞書内包表記114
 実引数19
 じゃんけんプログラム36
 集合75
 集合内包表記114
 縮小統治法162
 出力2
 条件式22, 98
 条件分岐3, 22, 98
 乗算13
 除算13
 しらみつぶし法162
 シングルクォート10
 数10, 62
 数学的帰納法127
 数値62
 スーパークラス153
 スコープ44, 117
 スタック87
 スタック・トレースバック48
 ステートメント17
 整数10
 撰氏20
 セッター151
 セル12, 15
 線形探索アルゴリズム5, 165
 全順序関係6
 全数探索法162
 選択ソート67, 169
 前置記法89
 挿入ソート68, 169
 ソーティング6, 67, 122
 ソーティングアルゴリズム169
 属性146
 素数45
 素数判定45

【た行】

大域変数44

ダイクストラの最短経路
 アルゴリズム82, 177
 代入文17
 代入文の処理系上の仕組み58
 多重グラフ77
 多重代入文17, 32, 71
 多相性66
 多態性66
 たぬき暗号69
 タプル12, 70
 ダブルクォート10
 タプル・パッキング12, 71
 探索問題5
 単純グラフ77
 誕生日当て問題168
 遅延評価60
 中間値の定理35
 抽象化158
 中置記法90
 中置表現66
 ツェラーの公式73
 停止性52
 データ型62
 データ管理テーブル58
 デコレータ140
 デフォルト値106, 120
 等価比較演算子12
 等差数列72
 到達可能性問題79
 動的計画法7, 104, 142, 163
 動的計画法とは178
 等比数列72
 特殊メソッド150
 特性関数112
 貪欲アルゴリズム7, 104, 169
 貪欲法162, 169

【な行】

内包的記法112
 ナップザック問題173
 名前17
 名前空間116
 名前渡し60

【英字】

args125
 assert 文51
 break110
 break 文110
 celebrity problem163
 continue110
 continue 文110
 else 文110
 except ブロック50
 for 文30, 107
 for ループ30, 107
 f 文字列10
 global 宣言117
 if 文98

二項演算子14
 二分木90
 二分探索アルゴリズム5, 167
 二分法34
 ニュートン・ラフソン法104
 入力2

【は行】

バケツによる水くみ問題184
 派生クラス153
 パッキング12
 バックトラック169
 バックトラック法162
 幅優先探索162
 半順序関係6
 半順序集合6
 番兵122
 回数14
 ビネの公式36, 131
 ビルトインスコープ44
 フィールド146
 フィボナッチ数列36, 129, 132
 ブール関数122
 ブール値10
 フェルマーの最終定理53
 フェルマーの小定理15
 深さ優先探索162
 複素数10
 不定方程式32, 184
 浮動小数点11
 浮動小数点数10
 プライベート化159
 フラット化96
 プログラミング2
 プログラム2, 3
 文17
 分割統治法122, 162
 分数計算149
 文脈自由文法134
 変数17, 58
 辺リスト77
 ボトムアップアプローチ182
 ポリモルフィズム66, 158

【ま行】

マージソート122, 169
 末尾再帰129
 水汲み問題33
 ミュータブル60
 無名関数定義21
 メソッド146
 メンバ147
 モジュール44
 モジュールスコープ44
 文字列68
 戻り値116
 問題解決162
 問題解決能力2

【や行】

山登り法162
 ユークリッドの互除法30, 72, 117, 127
 有限性52
 有向グラフ77
 有名人の問題163
 ユーロコイン170
 予約語17

【ら行】

ラウンドロビン・
 スケジューリング93
 ラムダ式21
 リスト66
 リスト探索問題5
 リスト内包表記112
 領域計算量54
 隣接行列77
 隣接リスト77
 ループ29
 例外処理47, 49
 例外発生51
 ローカルスコープ44
 ローカル変数44

ValueError49
 while 文29, 102
 while ループ29, 102
 ZeroDivisionError48

【ギリシャ文字】

β 規則21
 λ 計算21
 O 記法54
 k 乗根40
 α 変換44

【数字】

2 進数63
 8 進数63
 16 進数63

Jupyter Notebook12, 15
 kwargs125
 lambda21
 NameError48
 null オブジェクト10
 PEP122
 PEP821, 122
 print() 関数11
 range72
 sorted120
 stack traceback48
 swap71
 try ブロック49
 Turing 完全8
 turtle160
 TypeError49

— 著者略歴 —

富樫 敦 (とがし あつし)

1984年 東北大学大学院工学研究科博士後期3年の課程修了 (電気及通信工学専攻)

工学博士 (東北大学)

現在, 静岡理工科大学教授

コンピュータプログラミング

— Python でアルゴリズムを実装しながら問題解決を行う —

Computer Programming

— Problem Solving by Implementing Algorithms in Python —

© 一般社団法人 電子情報通信学会 2022

2022年4月28日 初版第1刷発行

検印省略

編者 一般社団法人
電子情報通信学会
<https://www.ieice.org/>

著者 富樫 敦
発行者 株式会社 コロナ社
代表者 牛来真也

印刷所 三美印刷株式会社
製本所 有限会社 愛千製本所

112-0011 東京都文京区千石 4-46-10

発行所 株式会社 コロナ社
CORONA PUBLISHING CO., LTD.

Tokyo Japan

振替 00140-8-14844・電話 (03)3941-3131(代)

ホームページ <https://www.coronasha.co.jp>

ISBN 978-4-339-01822-6 C3355 Printed in Japan



本書のコピー、スキャン、デジタル化等の無断複製・転載は著作権法上での例外を除き禁じられています。購入者以外の第三者による本書の電子データ化及び電子書籍化は、いかなる場合も認めていません。落丁・乱丁はお取替えいたします。