

はじめての SAGE

by Ted Kosan

翻訳:横田博史

Copyright © 2007 by Ted Kosan

This work is licensed under the Creative Commons Attribution-ShareAlike 3.0 License. To view a copy of this license, visit <http://creativecommons.org/licenses/by-sa/3.0/>

Table of Contents

1 はじめに	9
1.1 献辞.....	9
1.1 ご了承.....	9
1.2 支援団体.....	9
2 はじめに	10
2.1 数学計算環境とはなに?.....	10
2.2 SAGE はなに?.....	11
2.3 WEB サービスとして SAGE を利用.....	13
2.3.1 WEB サービスとして SAGE を利用 手法その1.....	14
2.4 SAGE のセルにソースコードを入力.....	17
3 SAGE でプログラムをする上での基礎	21
3.1 対象、変数、そして式.....	21
3.2 演算子.....	22
3.3 演算子の優先度.....	23
3.4 式で演算の順番を変更する事.....	24
3.5 変数.....	25
3.6 文.....	26
3.6.1 print 文.....	26
3.7 文字列(string).....	28
3.8 注釈(comment).....	28
3.9 条件演算子.....	29
3.10 if 文による判断.....	32
3.11 論理演算子 and,or,not.....	34
3.12 while 文による反復処理.....	35
3.13 長い反復処理, 無限反復処理, そして処理の中断.....	38
3.14 ワークシートのセルの挿入と削除.....	39
3.15 より高度な対象の型について.....	39
3.15.1 有理数.....	39
3.15.2 実数.....	40
3.15.3 他の対象の列を包含する対象: リストとタプル.....	41
3.15.3.1 多変数の取りまとめと取り崩し.....	42
3.16 リストとタプルを使った while 反復処理の利用.....	43
3.17 in 演算子.....	44
3.18 for 文を使った反復処理.....	44
3.19 関数.....	45
3.20 def 文を用いた関数の定義.....	45
3.21 SAGE 内部の関数の一部.....	47
3.22 SAGE 関数の情報を得る.....	55

3.23	利用者が入力した関数についても情報が使える.....	56
3.24	SAGE に含まれる関数を用いた例.....	56
3.25	for 文で srange()と zip()を利用.....	58
3.26	リスト内包(list comprehension).....	59
4	オブジェクト指向プログラミング.....	61
4.1	オブジェクト指向な心の書換.....	61
4.2	属性と振舞.....	61
4.3	類 (対象を生成する為の設計図).....	62
4.4	オブジェクト指向プログラムは必要に応じて対象の生成と破壊を行う.....	62
4.5	オブジェクト指向プログラム例.....	63
4.5.1	Hellos オブジェクト指向プログラム例 (無注釈).....	64
4.5.2	Hellos オブジェクト指向プログラム例 (注釈付き).....	65
4.6	SAGE 類(Class)と対象(Object).....	68
4.7	SAGE の対象について情報を得ること.....	69
4.8	対象のメソッドのリスト.....	71
4.9	継承による類の拡張.....	72
4.10	対象類、dir() 関数、組込メソッド.....	73
4.11	Sage.rings.integer.Integer 類の継承階層.....	75
4.12	"は一つの"関係.....	76
4.13	こんがらがったかな?.....	76
5	いろいろなこと.....	77
5.1	前回の処理結果の参照.....	77
5.2	例外処理.....	77
5.3	数値結果を得る.....	78
5.4	式の表記指南.....	79
5.5	組込定数.....	80
5.6	根.....	81
5.7	記号変数.....	81
5.8	記号変数.....	83
5.9	展開と因子分解.....	84
5.10	いろいろな記号式の例.....	84
5.11	記号式への値の引き渡し.....	85
5.12	記号方程式と solve()関数.....	85
5.13	記号数学関数.....	86
5.14	グラフを使った根の検出と find_root()メソッドを利用した数値的な根の検出.....	88
5.15	伝統的な書式で数学対象を表示.....	89
5.15.1	伝統的な数式で対象を表示する為に LaTeX を利用.....	89
5.16	集合.....	90
6	2D 描画.....	91
6.1	plot()と show()関数.....	91
6.1.1	描画の結合と描画の色の変更.....	93

6.1.2	グラフィックス対象とグラフィックスの結合.....	94
6.2	matplotlib による進んだ描画.....	96
6.2.1	網目と軸のラベルを持ったリストデータの描画.....	96
6.2.2	対数目盛の Y 軸を持った描画.....	97
6.2.3	描画の中にラベル付きの二つのグラフ.....	98
7	SAGE の書式.....	100
7.1	スピード書式.....	100
7.2	オープンオフィスプレゼン書式.....	100
8	高校数学の問題 (大半がまだ著作中).....	101
8.1	Pre-Algebra.....	101
8.1.1	方程式.....	101
8.1.2	式.....	101
8.1.3	幾何学.....	101
8.1.4	不等式.....	101
8.1.5	線形関数.....	101
8.1.6	Measurement.....	101
8.1.7	非線形方程式.....	102
8.1.8	Number Sense And Operations.....	102
8.1.8.1	分数の約分.....	102
8.1.9	多項式関数.....	103
8.2	代数.....	103
8.2.1	絶対値関数.....	103
8.2.2	複素数.....	103
8.2.3	合成関数.....	104
8.2.4	Conics.....	104
8.2.5	データ解析.....	104
9	離散数学: 初等的数とグラフ理論.....	104
9.1.1	方程式.....	104
9.1.1.1	記号分数の約分.....	104
9.1.1.2	二つの記号分数の積を計算.....	106
9.1.1.3	x について線型方程式を解く.....	107
9.1.1.4	分数を持つ線型方程式の解法.....	108
9.1.2	指数関数.....	110
9.1.3	冪.....	110
9.1.4	式.....	110
9.1.5	不等式.....	110
9.1.6	逆関数.....	110
9.1.7	線型方程式と関数.....	111
9.1.8	線型プログラミング.....	111
9.1.9	対数関数.....	111
9.1.10	兵站関数.....	111
9.1.11	行列.....	111
9.1.12	Parametric Equations.....	111
9.1.13	区分関数.....	111

9.1.14	多項式函数.....	112
9.1.15	冪級数函数.....	112
9.1.16	Quadratic Functions.....	112
9.1.17	Radical Functions.....	112
9.1.18	有理函数.....	112
9.1.19	列.....	112
9.1.20	級数.....	112
9.1.21	方程式系.....	113
9.1.22	変換.....	113
9.1.23	三角函数.....	113
9.2	Precalculus And Trigonometry.....	113
9.2.1	二項定理.....	113
9.2.2	複素数.....	113
9.2.3	合成函数.....	113
9.2.4	Conics.....	114
9.2.5	データ解析.....	114
10	離散数学: Elementary Number とグラフ理論.....	114
10.1.1	方程式.....	114
10.1.2	指数函数.....	114
10.1.3	逆函数.....	114
10.1.4	対数函数.....	114
10.1.5	兵站函数.....	114
10.1.6	行列と行列代数.....	115
10.1.7	数学的解析.....	115
10.1.8	Parametric Equations.....	115
10.1.9	区分函数.....	115
10.1.10	Polar Equations.....	115
10.1.11	多項式函数.....	115
10.1.12	冪関数.....	115
10.1.13	多項式函数.....	116
10.1.14	Radical Functions.....	116
10.1.15	有理函数.....	116
10.1.16	実数.....	116
10.1.17	列.....	116
10.1.18	級数.....	116
10.1.19	集合.....	116
10.1.20	方程式系.....	117
10.1.21	変換.....	117
10.1.22	三角函数.....	117
10.1.23	ベクトル.....	117
10.2	解析.....	117
10.2.1	微分.....	117
10.2.2	積分.....	117
10.2.3	極限.....	117
10.2.4	多項式近似と級数.....	118

10.3 統計.....	118
10.3.1 データ解析.....	118
10.3.2 Inferential Statistics.....	118
10.3.3 標準分布.....	118
10.3.4 1変数解析.....	118
10.3.5 確率と試行.....	118
10.3.6 2変数解析.....	119
11 高校生の科学の問題.....	120
11.1 物理学.....	120
11.1.1 原子物理.....	120
11.1.2 円運動.....	120
11.1.3 力学.....	120
11.1.4 電磁界.....	120
11.1.5 流体.....	120
11.1.6 運動学.....	121
11.1.7 光.....	121
11.1.8 光学.....	121
11.1.9 相対性理論.....	121
11.1.10 回転運動.....	121
11.1.11 音響.....	121
11.1.12 波.....	121
11.1.13 熱力学.....	121
11.1.14 仕事.....	122
11.1.15 エネルギー.....	122
11.1.16 モーメント.....	122
11.1.17 Boiling.....	122
11.1.18 浮力.....	122
11.1.19 Convection.....	122
11.1.20 密度.....	122
11.1.21 Diffusion.....	123
11.1.22 Freezing.....	123
11.1.23 摩擦.....	123
11.1.24 熱伝導.....	123
11.1.25 Insulation.....	123
11.1.26 Newton の法則.....	123
11.1.27 圧力.....	123
11.1.28 プーリー.....	124
12 計算の基礎.....	125
12.1 計算機ってなあに.....	125
12.2 記号(symbol)って何?.....	125
12.3 記号としてビットの並びを使う計算機.....	126
12.4 文脈的意味.....	130
12.5 変項.....	130

12.6 モデル.....	131
12.7 機械語.....	133
12.8 コンパイラとインタプリタ.....	136
12.9 アルゴリズム.....	137
12.1 計算.....	139
12.2 アルゴリズムの記録に図式が使える.....	141
12.3 1 から10までの数の総和を計算.....	141
12.4 数学計算系の数学の部分.....	143
13 SAGE サーバーの立ち上げ.....	144
13.1 インターネットに基づく技術への入門.....	144
13.1.1 複数の計算機はどの様にして互いに通信するの?.....	144
13.1.2 TCP/IP プロトコルについて.....	145
13.1.3 クライアントとサーバー.....	147
13.1.4 DHCP.....	147
13.1.5 DNS.....	148
13.1.6 プロセスとポート.....	149
13.1.7 良く知られたポート, 登録されたポートと動的ポート,.....	153
13.1.7.1 著名なポート (0 - 1023).....	154
13.1.7.2 登録されたポート (1024 - 49151).....	156
13.1.7.3 動的/個人ポート (49152 - 65535).....	156
13.1.8 SSH (Secure SHell) サービス.....	156
13.1.9 ネットワーク上の計算機間のファイルを複写する為の scp の利用について.....	157
13.2 SAGE の構成 (まだ).....	157
13.3 Linux に基づく SAGE 版.....	159
13.4 SAGE の VMware 仮想計算機版(大半の Windows 利用者向け).....	159

2 1 はじめに

3 1.1 献辞

4 この本は Steve Yegge と彼の blog” Math Every Day(数学徒然)” に捧げます
5 (<http://steve.yegge.googlepages.com/math-every-day>).

6 1.1 ご了承

7 次の方々からは、この本についての意見を頂きました (もし、このリストの中に貴方の名前
8 が抜け落ちていたら、どうか私宛 (ted.kosan at gmail.com)に電子メールを下さい:

9 Dave Dobbs

10 David Joyner

11 Greg Landweber

12 Jeremy Pedersen

13 William Stein

14 Steve Vonn

15 Joe Wetherell

16 1.2 支援団体

17 この本の面倒を見ているのは `sage-support` で、次で対処出来ます:

18 <http://groups.google.com/group/sage-support> . このグループに電子メールを送る時には
19 必ず表題に “[Newbies book]” と入れて下さい。

20 2 はじめに

21 SAGE は記号代数と数値計算の為のオープンソースの数学計算環境(MCE)です。数学計算環境
22 は複雑なので、それに習熟する為に莫大な時間と労力を必要とします。ひとつの数学計算環
23 境を使えるようにする為に費やされるこの莫大な労力は、それを学ぶ為に必要となる労力に
24 値します。初心者が SAGE 使いの専門家になるには時間が暫く掛かりますが、幸運な事に、
25 問題を解く為に SAGE の専門家になる必要はありません。

26 2.1 数学計算環境とはなに？

27 数学計算環境は幅広い数学計算アルゴリズムを自動処理する事の出来る計算プログラムの集
28 合です。計算アルゴリズムは数学の殆ど全ての領域に存在し、そして、新しいアルゴリズム
29 は日々発展し続けています。

30 1960年代より、膨大な数の数学計算機環境が生成され、そして、次野リストには最も人
31 気のある物が含まれています：

32 http://en.wikipedia.org/wiki/Comparison_of_computer_algebra_systems

33 幾つかの環境は高度に特殊化している一方で、ある物は汎用のものです。幾つかのものは伝
34 統的な書式（これは数学の教科書で見られるものです）を表示したり、入力出来たり、ある
35 物は伝統的な数式を表示出来ても、入力はテキストでなければならなかったり、それから、
36 幾つかは数式出力も入力もテキストだけが可能であったりします。

37 伝統的な数式とテキスト書式の違いの例として、ここでは伝統的な書式の式を挙げておきま
38 す：

$$A = x^2 + 4 \cdot h \cdot x$$

39 それから、テキスト書式で同じ数式を示しておきましょう：

40
$$A == x^2 + 4*h*x$$

41 殆どの数学計算環境はある種の数学向け高水準プログラム言語を保有しています。

42 これによって、その環境が有する数学のアルゴリズムをプログラムで利用出来る様になりま
43 す。幾つかのこれらの数学向けプログラム言語は、既存のプログラム言語の周囲で構築さ
44 れ、それらが動作する特殊な環境向けに構築されています。

45 ある数学計算環境は商用で購入しなければなりません、その他のものはオープンソースな
46 の自由に利用出来ます。双方の環境は似た核となる機能を持っていますが、他の領域では通
47 常異なっています。

48 商用の環境はオープンソース環境よりもより洗練されており、それらは往々にして GUI を持
49 ち、数式を伝統的な書式で入力したり操作する事が比較的容易になっています。その一方で
50 商用環境は欠点も持っています。一つの欠点はソフトウェアを保持する会社が撤退する機会
51 が常に付き纏い、それによって環境を将来利用出来なくなる恐れがある事です。別の欠点は
52 商用環境を拡張する事が出来ない事で、その環境のソースコードが利用者が利用出来る様
53 になっていないからです。

54 オープンソースの数学計算環境は通常 GUI を持っていませんが、それらのユーザーインター
55 フェイスは汎用性があり、その環境のソースコードを望む人に対しては何時でも使えます。
56 これは人が興味を持てば使える環境で、望むままに拡張も出来る事を意味するのです。

57 2.2 SAGE はなに？

58 SAGE はオープンソースの数学計算環境で、数式をテキスト形式で入力し、テキスト形式、あ
59 るいは伝統的な書式で表示するものです。大半の数学計算環境が自分自身を包含するもので
60 すが、SAGE は他の数学計算環境を包含する事で、それ自体があるアルゴリズムを提供する傘
61 の様な手法を取っています。この戦略によって SAGE は簡単に将来の需要を取り込む事が容易
62 な構造で、複数の数学計算環境の力を提供する事が出来るのです。

63 SAGE は強力でとても人気のある Python プログラム言語で記述されており、SAGE の利用者が
64 扱える数学指向のプログラム言語は Python の拡張です。これは SAGE の熟練利用者は Python
65 プログラムの熟練者でなければならないことを意味します。Python プログラム言語の知識が
66 まさに SAGE を上手に使いこなせるかどうかの指標となり、その SAGE の熟練度を決定する手
67 助けとなる利用者の Python の知識もあります (表 1 参照)。

水準	知識
SAGE 習熟者	Python も SAGE もよく分かっている
SAGE 知見者	Python は知っているけど、SAGE だけを少々かじった程度
SAGE 新米	Python を知らないけど、最低一つのプログラム言語を挙げる事が出来る
プログラム作成新米	計算機がどのように動作するか知らないし、一度もプログラムを組んだ事が無い

表 1: SAGE 利用者の経験水準

68 この本は SAGE 新米の為のものです。読者は最低でも一つのプログラム言語が挙げられるも
 69 の Python でのプログラム経験は皆無だと仮定しています(もし、貴方が計算機のプログラ
 70 ムがどのように動作するのかを再確認しておく必要があれば、この本の[計算の基礎](#)の節を通読
 71 すると良いでしょう。) この本は SAGE を使って問題を解くために十分な Python の事を教示
 72 します。貴方が SAGE 知見者となる為の手助けをしますが、SAGE 熟練者になる前に、この本
 73 に関連する本から Python を学ばなければなりません。

74 もし、貴方がプログラムの初心者であれば、この本はおそらく貴方にとって余りにも上級
 75 過ぎるかもしれません。私は *The Professor and Pat Programming*
 76 *Series*(<http://professorandpat.org>) という無料本を書いており、それらはプログラム初心
 77 者向けになっています。もし、貴方がプログラム初心者であって、SAGE を使って学ぶ事に興
 78 味があるのなら、*The Professor and Pat Programming* 本から進める事を第一にすべきで、
 79 それらを済ませてから、この本に戻ってくるべきです。

80 SAGE のウェブサイト (sagemath.org) には他の SAGE の情報源に従った SAGE の沢山の情報が
 81 あります。

82 2.3 WEB サービスとして SAGE を利用

83 SAGE はその構造上可能な限り、色々な方法に柔軟に対処可能です。SAGE の超新米さんは
84 WEB ブラウザから WEB サービスとして SAGE を使うことになるでしょう。SAGE の任意の複製は
85 この WEB サービスが行えるように設定されています。 図 2.1 に三つの SAGE の WEB サービス
86 の手法を示しておきましょう：

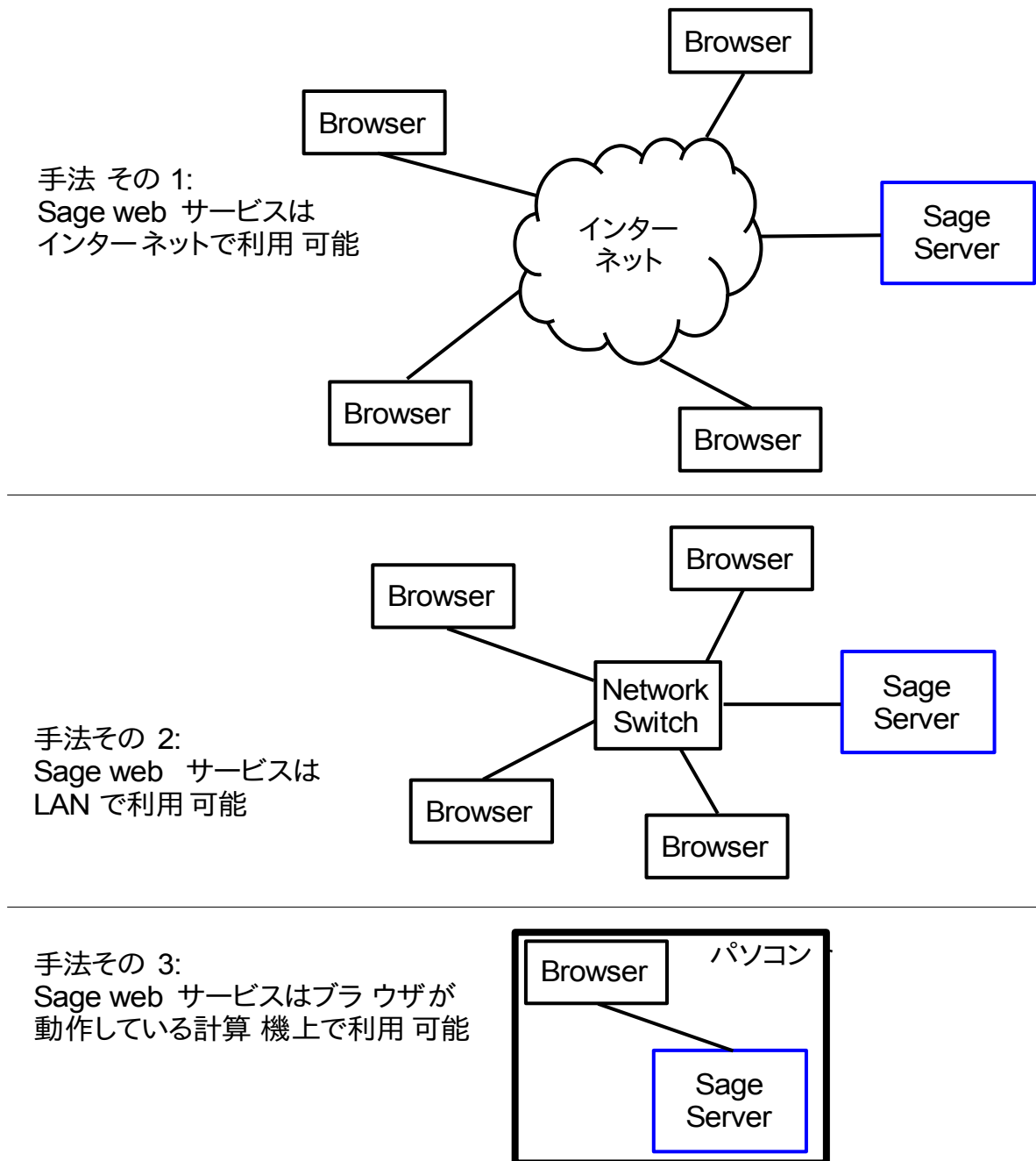


図 2.1: 三通りの Web サービスによるセッション

2.3.1 WEB サービスとして SAGE を利用 手法その 1

87 SAGE は現在 Firefox ウェブブラウザが動作に最適なので、もしも、貴方の計算機に Firefox
88 を入れていなければ、<http://mozilla.com/firefox>. で入手しておきましょう。

89 SAGE 開発チームは公開 SAGE ウェブサービスを(<http://sagenb.com>) で提供しており、この
90 サービスは SAGE のホームページのトップからも利用出来ます。この公開 SAGE ウェブサービ
91 ス向けのアカウントを取得する為には登録が必要なので、手順をここで解説しておきましょ
92 う。

93
94 Firefox ブラウザのウィンドウを開いて、下記を URL バーに書き込みます：

95 `http://sagenb.com`

96 すると Welcome ページが表示されます (図 2.2 を参照してください)

SAGE Mathematics Software: Welcome!

SAGE is a different approach to mathematics software.

The SAGE Notebook

With the SAGE Notebook anyone can create, collaborate on, and publish interactive worksheets. In a worksheet, one can write code using SAGE, Python, and other software included in SAGE.

General and Advanced Pure and Applied Mathematics

Use SAGE for studying calculus, elementary to very advanced number theory, cryptography, commutative algebra, group theory, graph theory, numerical and exact linear algebra, and more.

Use an Open Source Alternative

By using SAGE you help to support a viable open source alternative to Magma, Maple, Mathematica, and MATLAB. SAGE includes many high-quality open source math packages.

Use Most Mathematics Software from Within SAGE

SAGE makes it easy for you to use most mathematics software together. SAGE includes GAP, GP/PARI, Maxima, and Singular, and dozens of other open packages.

Use a Mainstream Programming Language

You work with SAGE using the highly regarded scripting language Python. You can write programs that combine serious mathematics with anything else.

Sign into the SAGE Notebook

Username:

Password:

[Sign up for a new SAGE Notebook account](#)

[Browse published SAGE worksheets
\(no login required\)](#)

図 2.2: SAGE Welcome screen.

97 SAGE ウェブサービスは SAGE ノート (Notebook) と呼ばれています。何故かという、数学

98 者が数学の計算を実行する為に伝統的に用いているノートを模擬しているからです。ノート
99 が使えるようになる前に、貴方はノートのアカウントの為に最初に登録をしなければなりま
100 せん。

101 **Sign up for a new SAGE Notebook account** リンクを選択すると登録ページが表示さ
102 れます(図 2.3 を参照して下さい)

Sign up for the SAGE Notebook.

Username:

Password:

Email Address:

[Cancel and return to the login page](#)

図 2.3: Signup page.

103 利用者名 (Username)とパスワード(Password)を Username と Password の入力欄に入れ、
104 それから **Register Now** ボタンを押します。すると、別のページが表示され、そこには登
105 録情報が受け取られた事と、承認メッセージが貴方が指定した電子メールアドレスに送られ
106 た事が書かれています。

107

108 このメールを開いて、その中に有るリンクを開きます。これで登録は完了し、いよいよ、貴
109 方はノートの **Welcome** ページに戻って、ログインが出来ます。

110 貴方のノートのアカウントでログインが成功すると、 **worksheet management** ページが
111 現れます (図 2.4 を参照して下さい)



図 2.4: *Worksheet management* ページ

112 物理的な数学ノートはワークシートを含んでいるので、SAGE の仮想ノートもワークシートを
 113 持っています。ワークシート管理ページはワークシートの生成、削除、インターネット上で
 114 印刷等の事が出来ます。これは新規に生成されたノートなので、未だ何もワークシートがあ
 115 りません。

116 **New Worksheet** リンクを選択すると新しいワークシートを生成します。ワークシートは伝
 117 統的な書式で数式を表示する為の特別な数学フォント、或いはこれらのフォントの画像が使
 118 えます。もし、貴方が作業している計算樹に数学フォントが入っていなければ、ワークシー
 119 トは代わりに組込みの画像フォントを用いる様に指示するメッセージが表示されます (図
 120 2.5 参照)



図 2.5: *jsMath* と *TeX* fonts が無い事の警告

121 画像フォントとは通常の数学フォントとしてはあまり明瞭ではありませんが、殆どの事には使
 122 えます。
 123 望めば、あとで数学フォントを貴方の計算機に入れられますが、ここでは取り敢えず **Hide**

- 124 **this Message** ボタンを押します。すると、真っ新のワークシートを含むページが現れます
125 (図 2.6 参照)



図 2.6: まっさらのワークシート

- 126 ワークシートは1つ、或いはそれ以上のセル(cell)を含みます。ここでセルは SAGE によって
127 実行されるソースコードに用います。セルは図 6 に示す様に長方形で、その中に沢山書込ん
128 でゆくにしがって大きくなります。ワークシートが最初に生成された時には最初のセルは
129 そのワークシートの上端にあり、これが貴方が通常式を書き込む場所になります。

130 2.4 SAGE のセルにソースコードを入力

- 131 早速、SAGE を簡単な計算樹として使って試してみましょう。貴方のマウスカールを貴方の
132 ワークシートの上端にあるセルの中に置いて下さい。カーソルが自動的に新しいセルの左端
133 に置かれた事に注意して下さい。貴方は SAGE のソースコードの新しい行をセルの左端から改
134 行を入れずに開始しなければなりません (貴方が他の事をするように指図されるまでは)。

- 135 次の式を入力しますが、Enter キーは押さないで:

136 $2 + 3$

- 137 貴方のワークシートは 図 2.7 の様になっている筈です。

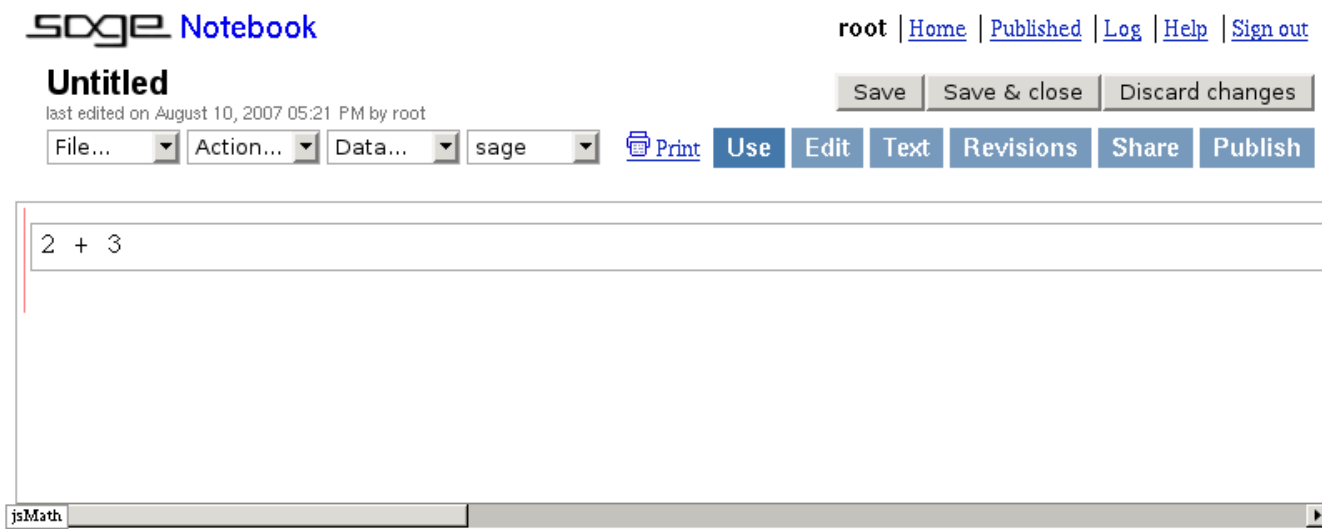


図 2.7: セルに式を入力

138 この時点で貴方には二つの選択肢があります。Enter キー<enter>を押すか、Shift キーを押
139 しながら Enter キー<shift><enter>を押すかの何れかです。単純に Enter キーを押せば、セル
140 は展開されてカーソルは次の行に移動して、ソースコードの入力が続けられます。

141 **Shift** と **Enter** を押すとワークシートはセルに入力されたすべてのソースコードを取り込んで
142 ネットワークを通じて SAGE サーバーにそのコードを**処理出来る様**に送り付けます。SAGE
143 に処理すべきソースコードが与えられると最初に SAGE 前処理機と呼ばれるソフトウェアを
144 用いた処理になります。この前処理機では SAGE ソースコードを Python ソースコードに変換
145 して SAGE を構築した Python 環境を用いて処理出来る様にします。

146 変換されたソースコードは Python 環境を通過しますが、そこで、**Python バイトコード**と
147 呼ばれる特殊な書式の機械言語に翻訳されます。このバイトコードはハードウェア CPU を模
148 擬するプログラムで処理されますが、このプログラムの事を **Python 翻訳機**と呼びます。

149 サーバがそのコードを効率良く処理出来る事もあれば、時間が掛かる事もあります。その
150 コードがサーバーで実行されている間、件のワークシートは小さな緑の縦棒を図 2.8. に示す
151 様にセルの下側、ウィンドウの左端に表示します。

The screenshot shows the SAGE Notebook interface. At the top, it says "SAGE Notebook" and "tkosan2 | Home | Published | Log | Help | Sign out". Below that, it says "Untitled" and "last edited on August 08, 2007 05:45 PM by tkosan2". There are buttons for "Save", "Save & close", and "Discard changes". Below these are dropdown menus for "File...", "Action...", "Data...", and "sage", along with a "Print" icon and buttons for "Use", "Edit", "Text", "Revisions", "Share", and "Publish". The main area contains a cell with the text "2 + 3" and a green vertical bar on the left side. An arrow points to the green bar with the text: "緑の縦棒は SAGE サーバが上のセルで <shft><enter> を押すことで投入したコードを現在処理中である事を指し示しています。"

図 2.8: セルの式を処理中

152 サーバーがソースコードの処理を終えると緑の棒が消えます。もし、表示可能な結果が生成
 153 されていれば、この結果は件のワークシートに返送され、ワークシートはその結果を式を投
 154 入したセルの真下の領域に表示します。

155 ここで shift と enter を貴方のセルで押して、ちょっと待てば 図 2.9 に示すような結果が得
 156 られます。

The screenshot shows the SAGE Notebook interface after the calculation. The main area contains a cell with the text "2 + 3" and the result "5" displayed below it. The green vertical bar is no longer present. The interface elements at the top are the same as in Figure 2.8.

図 2.9: 処理結果の表示

- 157 ノートの下セルから処理させるためにコードを投入すると、そのコードの処理をサーバー
158 が終わると自動的にこのセルの下に真っ新のセルが付加されます。
- 159 そこで、図 2.10 の 2 番目のセルにあるソースコードを入力して処理しましょう。

The screenshot shows the SAGE Notebook interface. At the top left is the logo "SAGE Notebook". To the right is the user name "root" and a navigation menu with links: Home, Published, Log, Help, Sign out. Below the logo is the title "Untitled" and a timestamp "last edited on August 10, 2007 05:21 PM by root". A toolbar contains buttons for "Save", "Save & close", "Discard changes", "Print", "Use", "Edit", "Text", "Revisions", "Share", and "Publish". Below the toolbar are dropdown menus for "File...", "Action...", "Data...", and "sage". The main workspace contains two cells. The first cell has the input $2 + 3$ and the output 5 . The second cell has the input $5 + 6 * 21 / 18 - 2^3$ and the output 4 . At the bottom left, a "jsMath" label is visible.

図 2.10: もっと複雑な処理

160 3 SAGE でプログラムをする上での基礎

161 3.1 対象、変数、そして式

162 ソースコード行

163 $2 + 3$

164 と

165 $5 + 6 * 21 / 18 - 2^3$

166 の双方を式(expression)と呼び、次を何が式であるかの定義とします:

167 プログラム言語での式(expression)とは、値、変数、演算子、関数の組合せであり、既存
168 の特定の規則や、それらを計算したり別の値で置き換える特定のプログラム言語の介在で解
169 釈(評価)されるものです。式を評価するとはその値の事を指します。数学で式はそれ自体
170 の評価された値です(あるいは持つと言っても良いでしょう);式はその値の表現です。
171 ([http://en.wikipedia.org/wiki/Expression_\(programming\)](http://en.wikipedia.org/wiki/Expression_(programming)))

172 計算機では、値 (value)は一つ、或いはそれ以上のメモリでのビットの並びで、与えられ
173 た文脈(context)を使って解釈される時の何者かを意味します。SAGE では意味を持ったメモ
174 リでのビットの並びを対象(object)と呼びます。SAGE 自体は対象で構成されており、その
175 SAGE プログラムが生成する与件もまた対象として表現されます。対象は第4章 でより詳細
176 に解説します。

177 上の式で、2, 3, 5, 6, 21, と 18 は対象で、`sage.rings.integer.Integer` 文脈と呼ばれ
178 る文脈を用いて解釈されます。文脈は 型(type)と呼ばれる対象に関連付けられ、型
179 `sage.rings.integer.Integer` は整数(integer)の表現で用いられる型の対象です。SAGE には
180 `type()`と呼ばれる命令があり、これは引数として与えた任意の対象の型を返します。`type()`
181 命令を使って、対象 3 と 21 の型が何であるか、次のコードを実行させてみましょう:(註:こ
182 の点からソースコードはセルに入れるべきで、さらに表示されるべき任意の結果は GUI を使
183 わないものにすべきです。)

```
184 type(3)
185 |
186 <type 'sage.rings.integer.Integer'>
```

```
187 type(21)
188 |
189 <type 'sage.rings.integer.Integer'>
```

190 対象の型の情報がなにであるかを type() に尋ねる方法は、'type' の右側の括弧の中にその対
191 象を置くことです。

192 3.2 演算子

193 上の式の文字+, -, *, /, ^ は**演算子**と呼ばれ、それらの目的は SAGE に式中の対象に対して
194 実行すべき操作が何であるかを伝える事です。例えば、式 2+3 では**和**の演算子+は SAGE に
195 整数 2 と 3 の和を計算し、その結果を返却すべき事を伝えています。ここで対象 2 と 3 は
196 sage.ring.integer.Integer 型なので、それらの和から得られた結果は共に
197 sage.rings.integer.Integer 型の対象になります。

198 **差**の演算子は-で、**積**の演算子は*, /は**商**の演算子で、%は**剰余**の演算子、そして、^は**冪**
199 の演算子です。SAGE はこれらに加え、他にも沢山の演算子を持っており、それらに関する
200 詳細な情報は Python の文書で見つけることが可能です。

201 次の例では、-, *, /, % と ^ 演算子が用いられています:

```
202 5 - 2
203 |
204 3
```

```
205 3*4
206 |
207 12
```

```
208 30/3
209 |
210 10
```

```
211 8%5
212 |
213 3
```

```
214 2^3
215 |
216 8
```

217 文字-は負の数を示す為にも用いられます:

```
218 -3
219 |
220 -3
```

221 負の数の差は正数になります:

```
222 - -3
223 |
224 3
```

225 3.3 演算子の優先度

226 式が一つ以上の演算子を含む場合、SAGE は**演算子の優先度**と呼ばれる規則の集合を用い
 227 て、その式の対象に作用させる演算子の順位を定めます。演算子はまた**演算子の順位**も参照
 228 します。より高い優先度の演算子は優先度の低い演算子よりも先に評価されます。次の表で
 229 は SAGE の優先度の規則の一部を示し、高い優先度の演算子が表の上に来る様にしています:

230 ^ 指数は右から左に評価されます。

231 *,%,/ 積、剰余、そして商演算子については左から右に評価されます。T

232 +, - 最後に、和と差は左から右に評価されます。

233 沢山の演算子を含む式を手早く扱える様に、これらの優先度の規則を実際に人力で試してみ
 234 ましょう。ここで、ソースコードの式は:

```
235 5 + 6*21/18 - 2^3
```

236 すると、伝統的な書式はこうなります:

$$5 + \frac{6 \cdot 21}{18} - 2^3$$

237 この優先度規則に従うと、これが式の演算子を SAGE が評価する際の順番になります:

```
238 5 + 6*21/18 - 2^3
239 5 + 6*21/18 - 8
240 5 + 126/18 - 8
241 5 + 7 - 8
242 12 - 8
243 4
```

244 最初の式から始めると SAGE はその下の式の 8 を結果とする ^ 演算子を最初に評価します。
245 2 番目の式では * 演算子が次に実行され、以降、順番に実行されて、最後の式は全ての演算子
246 が評価された後の最後の結果として 4 を示しています。

247 3.4 式で演算の順番を変更する事

248 式に含まれる演算子の順序は括弧の中に式の一部を纏める事で変更する事が可能です。括弧
249 の中に式を入れる事で他の演算子が評価されるよりも、その式が強制的に評価されます。例
250 えば、式 $2+4*5$ は通常 of 優先度規則を用いて評価すると 22 になります:

```
251 2 + 4*5  
252 |  
253     22
```

254 もし、 $4+5$ の回りに括弧を置くと、積の前に強制的に和の評価が行われるので結果は 30 にな
255 ります:

```
256 (2 + 4)*5  
257 |  
258     30
```

259 括弧は入れ子にする事が可能で、最も内側の括弧がその外側よりも先に評価されます:

```
260 ((2 + 4)*3)*5  
261 |  
262     90
```

263 括弧は他の任意の演算子よりも先に評価されるので、これらを優先度表の一番上に置いてお
264 きます:

265 () 括弧はその内側から先に評価される。

266 ^ 指数は右から左に評価される。

267 *, %, / 積、剰余、そして商演算子については左から右に評価されます。

268 +, - 最後に、和と差は左から右に評価されます。

269

3.5 変数

270 変数は人間が数の代わりにメモリの中のビットの羅列記号が参照出来る様にメモリの番地
271 に関連させられた名前です。SAGE で変数を生成する一つの方法は割当を通じて行う事で、
272 この場合は記号 '=' の左側に貴方が生成したい変数の名前を置き、記号の右側に式を置きま
273 す。式が対象を返すと対象はその変数に割当てられています。

274 次の例では、box という変数を生成し、数 7 をそれに割り当てます：

```
275 box = 7  
276 |
```

277 以前の例とは異なり、結果がワークシートに帰っていない事に注意してください。何故な
278 ら、その結果は変数 **box** に置かれたからです。box の内容を見なければ真つ新のセルにその
279 名前を入れてセルを評価します：

```
280 box  
281 |  
282 7
```

283 この例で見ることが出来る様にワークシート上の、とあるセルで生成された変数はワーク
284 シートの他のセルでも利用可能です。変数はワークシートが開かれている限り存在します
285 が、ワークシートを閉じると、その変数は失われます。ワークシートを再び開くと変数は割
286 当が行われたセルを評価する事で再び生成する必要があります。変数はワークシートを閉じ
287 る前に保存する事も可能で、ワークシートを再び開いた時に読込む事も可能ですが、これは
288 後で解説するより上級の項目になります。

289 SAGE 変数は大文字と小文字を区別します。これは二つ、あるいはそれ以上の変数名が同じ変
290 数かそうでないかどうかを決定する時に、SAGE が変数名に現れる個々の文字を調べ上げる事
291 を意味します。例えば、変数名 **Box** と変数名 **box** は同じ変数ではありません。何故なら、最
292 初の変数名は大文字 'B' で始まっていますが、二番目の変数名は小文字の 'b' で開始している
293 からです。

294 プログラムは 1 変数以上持つ事が可能です。ここでは 3 変数のより洗練した例を示します：

```
295 a = 2  
296 |  
  
297 b = 3  
298 |
```



```
299 a + b
300 |
301     5
```

```
302 answer = a + b
303 |
```

```
304 answer
305 |
306     5
```

307 記号 '=' の式の右側にある部分は常に最初に評価され、その結果は等号記号の左側の変数に
308 割り当てられます。

309 変数を `type()` 命令に通すと、変数に割り当てられた対象の型が返されます:

```
310 a = 4
311 type(a)
312 |
313     <type 'sage.rings.integer.Integer'>
```

314 与件型と `type` 命令は後でその詳細を解説します。

315 3.6 文

316 文は[アルゴリズム\(algorithm\)](#)の記述で用いられるプログラム言語の一部分です。式とは違っ
317 て、文は対象を返却せず、それらが作り出せる様々な効果の為に用いられます。文は式と
318 個々の文を含み、プログラムは文の列を用いて構成されます。

319 3.6.1 print 文

320 セルの一つ以上の式が表示可能な結果を生成すると、セルは頭の式からだけ結果を表示しま
321 す。例えば、このプログラムは3変数を生成し、これらの変数の内容を表示しようとしています:

```
322 a = 1
323 b = 2
324 c = 3
325 a
326 b
327 c
328 |
329     3
```

330 SAGE では、プログラムは一度にその一行を処理し、その際にコードの一番上の行の処理を開
331 始し、そして、そこから下側の処理を行います。この例では、 $a = 1$ の行が最初に実行され
332 ると $b = 2$ の行が実行され等々となります。しかしながら、全ての 3 変数に何が割当てられ
333 ているかを見たくても、最後の変数に割当てられた物だけが表示される事に注意してくださ
334 い。

335 SAGE は `print` と呼ばれる文を持ち、それはセルに置かれた式であれば、その結果を表示する
336 事を許容するものです。この例は前の例と一つの例外を除いて同様です。つまり、`print` 文
337 は全ての 3 変数を表示する為に用いられている点です：

```
338 a = 1
339 b = 2
340 c = 3
341 print a
342 print b
343 print c
344 |
345     1
346     2
347     3
```

348 `print` 文は、式と式の間にはコンマが置かれたものが引き渡されると、同じ行に複数の結果を
349 表示します：

```
350 a = 1
351 b = 2
352 c = 3*6
353 print a,b,c
354 |
355     1 2 18
```

356 コンマは `print` 文に引き渡される変数や対象の後ろに置かれると、`print` 文に表示を終えた
357 あとも次の行にカーソルを落とさない事を指示します。それ故に、`print` 文が実行されたあ
358 とには、その前の `print` 文の出力と同じ行にその出力が置かれます。

359 一つのセルから複数の結果を表示させる別の方法はセミコロン';'を使うことです。SAGE で

360 はセミコロンは文の末尾にオプションの行の末尾を示す文字として置く事が出来ますが、大
361 半は同じ行に複数の文を置く為だけに用いられています。次の例ではセミコロンは変数 a, b
362 と c を一行で初期化する為に用いられていることを示すものです:

```
363 a=1;b=2;c=3
364 print a,b,c
365 |
366     1 2 3
```

367 次の例は、セミコロンがセルからの複数の結果を出力する為にも、どのように用いられてい
368 るかを示すものです:

```
369 a = 1
370 b = 2
371 c = 3*6
372 a;b;c
373 |
374     1
375     2
376    18
```

377 3.7 文字列(string)

378 文字列(string)は文字に基づく情報を保つ為に用いる対象の型です。文字列の対象を生成
379 する為に使われる典型的な式は二重引用符 或いは単引用符 で囲まれた文書で出来ていま
380 す。文字列は丁度数の様に変数を用いて参照することが可能で、print 文を使って文字列を
381 表示する事も出来ます。次の例では文字列対象を変数 'a' に割り当て、文字列対象を 'a' を参
382 照する事で表示し、それから、その型も表示させています。

```
383 a = "Hello, I am a string."
384 print a
385 type(a)
386 |
387     Hello, I am a string.
388     <type 'str'>
```

389 3.8 注釈(comment)

390 ソースコードは往々にして理解し難いもので、それ故に、全てのプログラム言語ではコード
391 の中に注釈を書込めます。注釈はその近くにあるコードが何を遂行する物であることを説明す
392 る為に用いられ、ソースコードを眺めている人間が読むものと意味づけられています。注釈

393 はプログラムが実行される時には無視されます。

394 SAGE でソースコードに注釈を入れる方法が二つあります。第一の方法は記号'#'を任意の文
395 書の左側に置いて、サーバーに注釈である事を意味付けておきます。記号'#'から行末尾迄の
396 文書は注釈として扱われます。ここで、記号'#'を使った注釈を含むプログラムを示しておき
397 ます:

```
398 #これは注釈
399 x = 2 #変数 x に 2 を割り当て
400 print x
401 |
402     2
```

403 このプログラムを実行すると、記号'#'で開始する文書は無視されます。

404 SAGE プログラムに注釈を入れる第二の方法は3個の引用符で注釈を囲む事です。このオプ
405 ションは注釈が長過ぎて一行に収まらない時に便利です。このプログラムは3個の引用符を
406 用いた注釈を示しています:

```
407 """
408 これは長い注釈で一行よりも長いものに使います。
409 次のコードでは変数 x に 3 を割り当てて、x を印字します。
410 """
411 x = 3
412 print x
413 |
414     3
```

415 3.9 条件演算子

416 **条件演算子**は二つの対象を比較する為に用いられる演算子です。条件演算子を含む式は論理
417 値 (boolean) 対象を返し、論理値対象は True か False の何れか一つになります。表 2 に
418 SAGE で用いる条件演算子を示しておきます:

演算子	概要
<code>x == y</code>	二つの対象が同値であれば True を返し、それらが同値でなければ False を返します。==は比較を実行し、=の様な意味を持たないことに注意してください
<code>x <> y</code>	対象が同値でなければ True を返し、同値であれば False 。
<code>x != y</code>	同値でなければ True を返し、同値ならば False 。
<code>x < y</code>	左側の対象が右側よりも小さければ True を返し、左側の対象が右側よりも小さくなければ False 。
<code>x <= y</code>	左側の対象が右側の対象以下であれば True を返し、左側の対象が右側の対象以下でなければ False 。
<code>x > y</code>	左側の対象が右側の対象よりも大であれば True を返し、左側の対象が右側の対象よりも大でなければ False 。
<code>x >= y</code>	左側の対象が右側の対象以上であれば True を返し、左側の対象が右側の対象以上でなければ False 。

表 2: 条件演算子

419 次の例では、表 2 の各条件演算子を変数 `x` と `y` に置かれた対象の比較で用いられている様子
420 を示すものです:

```
421
422 # Example 1.
423 x = 2
424 y = 3

425 print x, "==", y, ":", x == y
426 print x, "<>", y, ":", x <> y
427 print x, "!=", y, ":", x != y
428 print x, "<", y, ":", x < y
429 print x, "<=", y, ":", x <= y
430 print x, ">", y, ":", x > y
431 print x, ">=", y, ":", x >= y
432 |
433     2 == 3 : False
434     2 <> 3 : True
435     2 != 3 : True
436     2 < 3 : True
437     2 <= 3 : True
438     2 > 3 : False
439     2 >= 3 : False

440 # Example 2.
```

```
441 x = 2
442 y = 2

443 print x, "==", y, ":", x == y
444 print x, "<>", y, ":", x <> y
445 print x, "!=", y, ":", x != y
446 print x, "<", y, ":", x < y
447 print x, "<=", y, ":", x <= y
448 print x, ">", y, ":", x > y
449 print x, ">=", y, ":", x >= y
450 |
451     2 == 2 : True
452     2 <> 2 : False
453     2 != 2 : False
454     2 < 2 : False
455     2 <= 2 : True
456     2 > 2 : False
457     2 >= 2 : True

458 # Example 3.
459 x = 3
460 y = 2

461 print x, "==", y, ":", x == y
462 print x, "<>", y, ":", x <> y
463 print x, "!=", y, ":", x != y
464 print x, "<", y, ":", x < y
465 print x, "<=", y, ":", x <= y
466 print x, ">", y, ":", x > y
467 print x, ">=", y, ":", x >= y
468 |
469     3 == 2 : False
470     3 <> 2 : True
471     3 != 2 : True
472     3 < 2 : False
473     3 <= 2 : False
474     3 > 2 : True
475     3 >= 2 : True
```

476 条件演算子は今迄解説してきた他の演算子と比較して低い優先度に置かれています:

477 () 括弧はその内側から先に評価される。

478 ^ 指数は右から左に評価される。

479 `*,%,/` 積、剰余、そして商演算子については左から右に評価されます。

480 `+, -` それから和と差は左から右に評価されます。

481 `==, <>, !=, <, <=, >, >=` 最後に条件演算子が評価されます。

482 3.10 if 文による判断

483 全てのプログラム言語には判断を行う機能が提供されており、SAGE で最も一般的に用いられ
484 る判断を行う文として if 文があります。

485 If 文の単純化した構文は次のようになります:

```
486 if <式>:
```

```
487     <文>
```

```
488     <文>
```

```
489     <文>
```

```
490     .
```

```
491     .
```

```
492     .
```

493 if 文の動作は、その if のすぐ右に置かれた式を評価して、それから、返却された対象を調
494 べます。ここで、この対象が”真”であれば if 文内部の文が実行されます。対象が”偽”で
495 あれば、if 文内部の文は実行されません。

496 SAGE では対象が**非零**、或いは**非空**であれば”真”で、**零**、あるいは**空**になれば”偽”となり
497 ます。一つ、あるいはそれ以上の条件演算子を含む式は**論理値(boolean)**対象の **True** か
498 **False** を返却します。

499 if 文内部に文を置く方法は、if 文の先頭部分の末尾にコロン’`:`’を置いて、一つ、あるいは
500 それ以上の文をその下に置きます。if 文内側に置く文は、if 文の行に続けて、一つあるいは
501 それ以上の **tab** や **space** を使って、if 文の行の左側の先頭から字下(indent)をしていなく
502 ればなりません。その上、全ての字下をされた文は同じ方法で、しかも、同じ分量だけ字下
503 を行っていなければなりません。この様に字下げされた一つ、あるいはそれ以上の文はコー
504 ドの一塊として参照されます。

505 次のプログラムでは if 文を変数 `x` に割り当てられた数値が 5 よりも大であるかどうかを決定
506 させる為に用いています。もし、`x` が 5 よりも大であれば、プログラムは”Greater”を表示

507 し、それから”End of program”を表示します。

```
508 x = 6
```

```
509 print x > 5
```

```
510 if x > 5:
511     print x
512     print "Greater"
```

```
513 print "End of program"
514 |
515     True
516     6
517     Greater
518     End of program
```

519 このプログラムの x は 6 が割り当てられており、それ故に $x > 5$ は真となります。この式が
520 表示された時、6 が 5 よりも大なので論理値対象 True を表示します。

521 `if` 文で式を評価し、それが **True** の時、`if` 文はその内側の `print` 文を実行し、それから変数
522 x に割り当てられた対象を文字列”Greater”の前に表示します。もし、追加の文が `if` 文の内
523 側に置く必要があれば、その追加の文は `print` 文と同じ字下位置で、その下に追加します。

524 最後に、終に表示した文字列”End of program”は `if` 文が何をしようと無関係です。

525 ここで 6 の代わりに x が 4 を割り当てられていた場合は：

```
526 x = 4
```

```
527 print x > 5
```

```
528 if x > 5:
529     print x
530     print "Greater."
```

```
531 print "End of program."
532 |
533     False
534     End of program.
```

535 ここで、式 $x > 4$ は対象 False を返し、これは `if` 文がその内側の文の実行を行わない原因
536 となります。

537 **3.11 論理演算子 and, or, not**

538 二つ、あるいはそれ以上の式が全て真であるかどうかを検証したければ **and** 演算子を使いま
539 しょう :

```
540 a = 7
541 b = 9
542 print a < 5 and b < 10
543 print a > 5 and b > 10
544 print a < 5 and b > 10
545 print a > 5 and b < 10
546 if a > 5 and b < 10:
547     print "These expressions are both true."
548 |
549     False
550     False
551     False
552     True
553     These expressions are both true.
```

554 また一群の式のうち、少なくとも一つが真であるかどうかを決定したければ **or** 演算子を使
555 いましょう :

```
556 a = 7
557 b = 9
558 print a < 5 or b < 10
559 print a > 5 or b > 10
560 print a > 5 or b < 10
561 print a < 5 or b > 10

562 if a < 5 or b < 10:
563     print "At least one of these expressions is true."
564 |
565     True
566     True
567     True
568     False
569     At least one of these expressions is true.
```

570 最後に、**not** 演算子で True の結果を False に、あるいは False の結果を True に変更すること
571 に使えます :

```
572 a = 7
573 print a > 5
574 print not a > 5
```

```
575 |
576     True
577     False
```

578 ブール演算子は他の演算子と比較してより低い優先度に行っています：

579 () 括弧は中から外へと評価します。

580 ^ それから冪は右から左と評価します。

581 *,%,/ それから、積、剰余、商演算子が左から右へと評価されます。

582 +, - そして、加法と減算が左から右へと評価されます。

583 ==, <>, !=, <, <=, >, >= それから条件演算子が評価されます。

584 not ブール演算子が最後に評価されます。

585 and

586 or

587 3.12 while 文による反復処理

588 計算機を含め、多彩な機械は繰り返される一連の動作という原理から、それらの力の大半を
589 導き出しています。

590 SAGE はプログラムに於ける反復処理を実装する為に、いくつかの手法を提供しますが、それ
591 らの手法は単刀直入のものからまどろっこしいものまであります。

592 単刀直入的な **while** 文から開始する事で、SAGE での反復処理の解説を始めましょう

593 While 文の構文指定は次のようなものです：

```
594 while <式>:
595     <文>
596     <文>
597     <文>
598     .
```

599 .
600 .

601 while 文は if 文に似ていますが、その頭の右側にある式が真である限り、その while 文が含
602 む文を反復して実行する事が異なります。その式が False の値を持つ対象になると、 **while**
603 文は包含する文の実行を飛ばして **while** 文に続く文 (存在していれば) の処理を継続して実
604 行します。

605
606 次の例題は while を用いた 1 から 10 までの整数を表示する反復処理です:

```
607 # 1 から 10 までの整数を表示
608 x = 1 #Initialize a counting variable to 1 outside of the loop.
609 while x <= 10:
610     print x
611     x = x + 1 #Increment x by 1.
612 |
613     1
614     2
615     3
616     4
617     5
618     6
619     7
620     8
621     9
622     10
```

623 このプログラムでは、 **x** という名前の一つの変数が生成されます。その変数は **print** 文に
624 どの整数を印字すべきかを伝え、そしてまた **while** による反復処理を継続すべきかどうかを
625 決定することにも用いられます。

626 プログラムが実行されると、1 は **x** に入れられて、それから **while** 文に入ります。式 $x \leq 10$
627 は $1 \leq 10$ になって、1 が 10 以下なので論理対象が True を包含しているとその式を繰り返しま
628 します。

629 while 文はその式が真となる対象が返されるので、その文に含まれるすべての文を実行しま
630 す。

631 print 文でその時点の **x** に含まれる値(それは 1) を印字し、それから $x=x+1$ が実行されま

632 す。

633 式 $x=x+1$ は多くのプログラミング言語で用いられる標準的な式です。この書式の式が評価さ
634 れる度に、変数を含む値に 1 を加えます。この式の x に与える効果は別途、 x に 1 増分すると
635 も言います。

636 この場合では、 x は 1 を包含し、その式の評価の後には x は 2 を包含します。

637 **while** 文内部の最後の文が実行されると、**while** 文は **while** の左側にある式の再評価を行っ
638 て、反復を継続するかどうかを決定します。この時点では x が 2 なので、その式は **True** を
639 返すので、コード内部の **while** 文は再び実行されます。この反復処理は x が 1 1 に達して、
640 その式が **False** を返すまで実行されます。

641 このプログラムは別の結果が得られる様に調整する事ができます。例えば、次のプログラ
642 ムは 1 から 1 0 0 迄の整数を表示する様に **while** の右にある 10 を 1 0 0 に書き換えたもので
643 す。そして、コンマを出力がウィンドウの右側に当たるまで同じ行に表示される様に **print**
644 文の後ろに置きます。

645 # 1 から 1 0 0 迄の整数を印字

646 $x = 1$

647 **while** $x \leq 100$:

648 **print** x ,

649 $x = x + 1$ #Increment x by 1.

650 |

651 | 1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26 27

652 | 28 29 30 31 32 33 34 35 36 37 38 39 40 41 42 43 44 45 46 47 48 49 50 51

653 | 52 53 54 55 56 57 58 59 60 61 62 63 64 65 66 67 68 69 70 71 72 73 74 75

654 | 76 77 78 79 80 81 82 83 84 85 86 87 88 89 90 91 92 93 94 95 96 97 98 99

655 | 100

656 次のプログラムは増分を 1 から 2 に変更する事で 1 から 9 9 迄の奇数を印字します:

657 #1 から 9 9 迄の奇数を印字

658 $x = 1$

659 **while** $x \leq 100$:

660 **print** x ,

661 $x = x + 2$ #Increment x by 2.

662 |

```
663 1 3 5 7 9 11 13 15 17 19 21 23 25 27 29 31 33 35 37 39 41 43 45 47 49 51
664 53 55 57 59 61 63 65 67 69 71 73 75 77 79 81 83 85 87 89 91 93 95 97 99
```

665 最後に、このプログラムは1から100迄の数を逆の順序で印字します:

```
666 # 1から100迄の整数を逆順で印字
```

```
667 x = 100
```

```
668 while x >= 1:
```

```
669     print x,
```

```
670     x = x - 1 #Decrement x by 1.
```

```
671 |
672 100 99 98 97 96 95 94 93 92 91 90 89 88 87 86 85 84 83 82 81 80 79 78 77
673 76 75 74 73 72 71 70 69 68 67 66 65 64 63 62 61 60 59 58 57 56 55 54 53
674 52 51 50 49 48 47 46 45 44 43 42 41 40 39 38 37 36 35 34 33 32 31 30 29
675 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2
676 1
```

677 この結果に達する為に、xの初期値を100にするので、反復処理を継続する為にxが1以
678 上(x>=1)である事を検証し、1を加えるのではなくxから1を引く様にしています。

679 3.13 長い反復処理，無限反復処理，そして処理の中断

680 膨大な時間を費やしたり、更には、無限の時間を費やす様な反復処理を行わせる事は、まっ
681 とうな事でも錯誤からであったも簡単な事です。無限反復処理を含むプログラムを実行して
682 しまったとき、

683 SAGEにその実行を中断する事を伝えない限り処理が継続します。これはワークシートの左上
684 側のActionメニューを選択してInterruptメニュー項目を選択する事で行われます。長
685 く実行されている反復処理を持つプログラムはこの方法でも中断する事ができます。双方
686 で、緑の縦棒がプログラムが実行中である事を示しており、プログラムを中断すれば緑の棒
687 が消えます。

688 このプログラムは無限反復処理を包含します:

```
689 #無限反復処理の例
```

```
690 x = 1
```

```
691 while x < 10:
```

```
692     answer = x + 1
```

```
693 |
```

694 反復処理内部の x の包含する値が決して変化しないので、式 $x < 10$ は常に True と判断される
695 ので、反復処理が継続するのです。

696 このプログラムを実行し、ワークシートの `Interrupt` 命令を使って中断します。時には
697 ワークシートが
698 実行を止めるのに十分でない事があれば、`Action->Restart worksheet` を選択しましよ
699 う。

700 ワークシートを再起動する時には、**全ての変数とその初期状態に戻されるので**、それらの変
701 数に値を指定したセルを再び実行する必要があるでしょう。

702 3.14 ワークシートのセルの挿入と削除

703 新しいワークシートのセルが必要になると、貴方のマウスカーソルをセルを挿入する上下の
704 二つのセルの間に移動すると**水平の青色の棒**が現れます。この青い棒を押すと新しいセル
705 がワークシートのその箇所に挿入されます。

706 もしもセルを削除したければ、セルのすべての文書を削除して、それを空にします。カーソ
707 ルを空のセルに置いたことを確認してからキーボードのバックスペースキーを押します。す
708 るとセルが削除されます。

709 3.15 より高度な対象の型について

710 この点について、我々は単に `'sage.rings.integer.Integer'` と `'str'` という対象の型しか扱っ
711 ていません。しかし、SAGE は多様な目的に応じて使える莫大な数の数学的、或いは非数学的
712 対象の型を保有します。次の節では、新たに二つの数学的対象と二つの非数学的対象を紹介
713 しましょう。

3.15.1 有理数

714 有理数は `sage.rings.rational.Rational` 型の対象に属します。次の例で、有理数 $1/2$
715 の型を表示し、変数 x に $1/2$ を割り当て、それから x が参照する対象の型を表示させていま
716 す:

```
717 print type(1/2)
718 x = 1/2
719 print x
720 type(x)
721 |
722     <type 'sage.rings.rational.Rational'>
723     1/2
724     <type 'sage.rings.rational.Rational'>
```

725 次のプログラムは前のプログラムを実行した後に、そのワークシートの別のセルに入力した
726 ものです。二つの有理数同士の和とその結果を示すもので、結果も有理数で、変数 y に割り
727 当てられています:

```
728 y = x + 3/4
729 print y
730 type(y)
731 |
732     5/4
733     <type 'sage.rings.rational.Rational'>
```

734 有理数と整数の和を行うと、その結果は `sage.rings.rational.Rational` 型の対象になりま
735 す。

```
736 x = 1 + 1/2
737 print x
738 type(x)
739 |
740     3/2
741     <type 'sage.rings.rational.Rational'>
```

3.15.2 実数

742 実数は `sage.rings.real_mpfr.RealNumber` 型の対象に属します。次の例題では、実
743 数.5の型を表示し、変数 x に.5を割り当てて変数 x を表示し、それから x が参照する対象の
744 型を表示させています:

```
745 print type(.5)
746 x = .5
747 print x
748 type(x)
749 |
750     <type 'sage.rings.real_mpfr.RealNumber'>
751     0.5000000000000000
752     <type 'sage.rings.real_mpfr.RealNumber'>
```

753 次の処理は前の処理を実行した後で、そのワークシートの別のセルに入力したものです。二
754 つの実数同士の和とその結果を示し、その結果もまた実数で、変数 y に割り当てています:

```
755 y = x + .75
756 print y
757 type(y)
758 |
```

```
759 1.2500000000000000
760 <type 'sage.rings.real_mpfr.RealNumber'>
```

761 実数と有理数との和により、その結果は `sage.rings.real_mpfr.RealNumber` 型の対象となり
762 ます:

```
763 x = 1/2 + .75
764 print x
765 type(x)
766 |
767 1.2500000000000000
768 <type 'sage.rings.real_mpfr.RealNumber'>
```

3.15.3 他の対象の列を包含する対象: リストとタプル

769 `list` 対象の型は順序がある対象、即ち、**列(sequence)**を保持するために設計されていま
770 す。`List` はとても柔軟性がある、SAGE で最も頻繁に用いられる型の一つです。`list` は任
771 意の型の対象を保持する事が可能で、必要に応じて大きくしたり小さくしたりする事が出来
772 れば、入れ子にする事も出来ます。`list` に含まれる対象は `list` に於けるそれらの位置で取
773 り出す事や他の対象で置き換える事も出来ます。`list` を大きくしたり小さくしたりする事
774 や、その包含する内容を変えられる能力により**可変(mutable)**対象型となります。大括弧の
775 対の替わりに `0` あるいはそれ以上の対象や式を置く事でリストを生成する手法があります。
776 次のプログラムではリストの型を印字する事から開始します。それから、それは数 `50`, `51`,
777 `52` と `53` を含むリストを生成し、変数 `x` に割り当てて `x` を印字します。次に `0` と `3` の位置にあ
778 る対象を印字し、`3` の位置にある `53` を `100` で置き換えて `x` を再度印字し、それから最後に `x`
779 の対象の型を印字します:

```
780 print type([])
781 x = [50, 51, 52, 53]
782 print x
783 print x[0]
784 print x[3]
785 x[3] = 100
786 print x
787 type(x)
788 |
789 <type 'list'>
790 [50, 51, 52, 53]
791 50
792 53
793 [50, 51, 52, 100]
794 <type 'list'>
```


795 リストの最初の対象が1ではなく0の位置であり、これによってリストの最後の対象の位置
796 はそのリストの長さよりも一つ小さくなる事に注意しましょう。また、変数の右側に位置番
797 号を大括弧で括れば、リストの中の対象を参照することになります。

798 次の例では異なった型の対象がリストの中に置ける事を示します：

```
799 x = [1, 1/2, .75, 'Hello', [50,51,52,53]]
800 print x
801 |
802     [1, 1/2, 0.7500000000000000, 'Hello', [50, 51, 52, 53]]
```

803 タプル (Tuples)もまた列(sequences)で、不可変であることを除いてリストに似ていま
804 す。タプルは大括弧ではなく小括弧を使うことで生成され、不可変(immutable)であると言
805 う事は、一旦、タプル対象を生成すると、大きく、小さくしたり、それが包含する対象を変
806 えられない事を意味します。次のプログラムは最初のリストプログラムに似ていますが、リ
807 ストの代わりにタプルを用いており、位置4の対象を変更しようとせず、print文の代わり
808 にセミコロンを使う手法で複数の結果を表示させています：

```
809 print type(())
810 x = (50,51,52,53)
811 x;x[0];x[3];x;type(x)
812 |
813     <type 'tuple'>
814     (50, 51, 52, 53)
815     50
816     53
817     (50, 51, 52, 53)
818     <type 'tuple'>
```

3.15.3.1 多変数の取りまとめと取り崩し

819 コンマで分離された多変数が一つの変数として割り当てられると、その変数は自動的にタプ
820 ルとして置かれ、これをタプル纏め(tuple packing)と呼びます：

```
821 t = 1,2
822 t
823 |
824     (1, 2)
```

825 タプルにコンマで区切った複数の変数を割り当てる時、これを**タプル展開(tuple**
826 **unpacking)**と呼びます：

```
827 a, b, c = (1, 2, 3)
828 a; b; c
829 |
830     1
831     2
832     3
```

833 タプル展開をする為にはタプル内部の対象の数は等号左側にある変数の総数と一致しなければ
834 なりません。

835 3.16 リストとタプルを使った while 反復処理の利用

836 反復処理を行う文はリストやタプルの各成分を次々と選択して、処理がそれらの対象に対し
837 て処理されるようにすることが出来ます。次のプログラムはリストの各成分を表示する為に
838 while 文による反復処理を使っています：

```
839 #リストの各成分を印字
840 x = [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
841 y = 0
842 while y <= 9:
843     print x[y]
844     y = y + 1
845 |
846     50
847     51
848     52
849     53
850     54
851     55
852     56
853     57
854     58
855     59
```

856 反復処理はまたリストでの検索にも使えます。次のプログラムでは **while** による反復処理と
857 **if** 文を使って、数 53 を含むかどうかリストの検索で用います。もし、53 をリストの中で見

858 つけると、メッセージを表示します。

```
859 # 53 がリストにあるかどうかを判断。
860 x = [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
861 y = 0
862 while y <= 9:
863     if x[y] == 53:
864         print "53 was found in the list at position", y
865     y = y + 1
866 |
867     53 was found in the list at position 3
```

868 3.17 in 演算子

869 反復処理は非常に便利な機能であり、内部的な反復を行う **in** と名付けられた演算子さえも
870 SAGE は持っています。 **in** 演算子はリストに与えられた対象が含まれているかどうかを判断
871 する為に自動的に検索を行うことができます。もし、その対象を見つけると、**True** を返却し
872 ますが、対象が見つからなければ **False** を返します。次のプログラムは両方の場合を示して
873 います:

```
874 print 53 in [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
875 print 75 in [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
876 |
877     True
878     False
```

879 **not** 演算子はその結果を変更する為に **in** 演算子と併用する事も出来ます:

```
880 print 53 not in [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
881 print 75 not in [50, 51, 52, 53, 54, 55, 56, 57, 58, 59]
882 |
883     False
884     True
```

885 3.18 for 文を使った反復処理

886 For 文は while 文でやっている様に、リストやタプルからの添字を用いた反復を使います
887 が、もっと柔軟で自律的です。ここでは for 文の簡略化した構文を示しておきます:

888 for <target> in <対象>:

```
889 <文>
890 <文>
891 <文>
892 .
893 .
894 .
```

895 この構文で、<target> は通常変数であり、<対象>は通常、複数の対象を包含する一つの対象
896 です。この節の残りでは、<対象>はリストであると仮定しておきます。for 文は交互にその
897 リストの対象を選出し、<target>に割り当て、インデントされた箇所の内部の文を実行しま
898 す。次のプログラムでは for 文をリスト内部の項目のすべてを表示する為に用いています：

```
899 for x in [50,51,52,53,54,55,56,57,58,59]:
900     print x
901 |
902     50
903     51
904     52
905     53
906     54
907     55
908     56
909     57
910     58
911     59
```

912 3.19 関数

913 プログラムした **Function** は、コードの名付けられたブロックで構成されていて、一つ、あ
914 るいは何回もそのプログラムの他の部分から呼び出されることによって実行され得る文で
915 す。関数は呼び出したコードから引き渡される対象を持ち、関数は呼び出したコードに対象
916 を返します。関数の例として、対象の型を決定する事に使える `type()`命令を挙げておきま
917 す。

918 関数は SAGE がコードを再利用できる一つの手法です。大半のプログラム言語では、この方法
919 でコードが用いられませんが、再利用される文のコードの型はサブルーチン、あるいは手続
920 と呼ばれています。関数名はすべての小文字を使います。もし、関数名が一つの語(例え
921 ば、`calculatesum`の様に)を含んでいれば、下線を語と語の間に置いて(`calculate_sum`のよ
922 うに)読み易くする事が出来ます。

923 3.20 def 文を用いた関数の定義

924 関数定義で用いられる文は `def` と呼ばれ、その構文は次の通りです：

```
925 def <関数名>(arg1, arg2, ... argN):
926     <文>
927     <文>
928     <文>
929     .
930     .
931     .
```

932 `def` 文は引数を伴った関数名を包含するヘッダを含みます。ここで引数は省略が可能です。
933 関数は 0、又はそれ以上の引数を持つことが可能で、それらの引数は括弧の中に置かれま
934 す。関数が呼び出されたときに実行される文はインデント付けられたプログラムブロックを使
935 った関数の内部に置かれます。

936 次のプログラムは `addnums` と呼ばれる関数を定義し、この関数は二つの数を引数として取
937 り、それらを足し合わせ、それから `return` 文を使って輪を返します：

```
938 def addnums(num1, num2):
939     """
940     num1 と num2 の和を返す
941     """
942     answer = num1 + num2
943     return answer
```

```
944 #関数を呼び出し、2に3を加えさせるC
945 a = addnums(2, 3)
946 print a
```

```
947 #関数を呼び出して4に5を加えさせる
948 b = addnums(4, 5)
949 print b
950 |
951 5
952 9
```

953 最初にこの関数を呼び出すと、数字の 2 と 3 が引き渡されて、これらの数は変数 `num1` と
954 `num2` にそれぞれ割り当てられます。関数の呼び出しの間に対象が引き渡された引数変数は必
955 要に応じて関数内部で利用することが出来ます。

956 関数がそれを呼び出したものに結果を返却する時に、`return` 文の右側に置かれた対象は呼
957 び出したプログラムの内部で利用出来ることに注意してください。関数自体が、その返却す
958 る対象で置き換えられるかのようです。返却された対象について別の見方は、呼び出した
959 コードの中で関数名の左側に送り出され、しかしながら、符号は一致し、その変数に割当て
960 られるというものです。最初の関数呼出では、関数が返す対象は変数' a 'に割り当てられたも
961 ので、この対象が印字されています。第 2 の関数呼出は最初の関数呼出に似ていますが、異
962 なった数(4, 5)を関数に与えている点が異なります。

963 3.21 SAGE 内部の関数の一部

964 SAGE は幅広く多様な目的に対応出来る様に莫大な数の組込み関数があります。

965 表 3 にはこれらの関数の一部を示しておきますが、SAGE の文書にはもっと長い関数の一覧
966 があります。より完全な関数の一覧は [SAGE Reference Manual](#) にあります。

Function Name	Description
abs	引数の絶対値を返却
acos	逆余弦函数.
add	数列(文字列ではない)と媒介変数' start' の値の総和を返します。列が空であれば start の値を返却します。
additive_order	x の additive order を返す。
asin	逆正弦函数。
atan	逆正接函数。
binomial	二項係数を返す函数。
ceil	足切り函数
combinations	複合集合(同時に同じ対象を含んでいるかもしれない対象のリスト)の組合せの mset は反復なしで順序付けられていない選出であり、mset のソートされた副リストによって表現されています。k 成分の複合集合 mset の全ての組合せの集合を返却します。
complex	実部とオプションの虚部から複素数を生成します。つまり(実部 + 虚部*1j) と同値で、実部の附置は 0 です。
cos	正弦函数
cosh	双曲余弦函数
coth	双曲余接函数
csch	双曲余割函数
denominator	x の分母
derivative	f の微分
det	x の行列式
diff	f の微分
dir	与えられた対象の(幾つかの)属性込みの名前や対象から到達可能な属性の一覧をアルファベット順で返します。
divisors	全ての整数因子を返す
dumps	文字列 s に対象 obj を吐き出す。対象 obj の復元には load(s)を使います。
e	自然底
eratosthenes	素数 <= n のリストを返す。
exists	S が P(x) が True となる成分 x を含む場合、この函数は True と成分 x を返し、それ以外は False と None を返します
exp	指数函数, $\exp(x) = e^x$.
expand	多項式を展開した書式で返す

factor	整数の因数分解をタプルのリストとして返します
factorial	n の階乗を計算します。階乗は積 $1 * 2 * 3 \dots (n-1) n$ です。
fibonacci	n 番目の Fibonacci 数を返却。
fibonacci_sequence	$n=start$ から $n=stop$ 迄の全ての fibonacci 数 f_n に対して、Fibonacci 列の反復を返却します。
fibonacci_xrange	与えた範囲で Fibonacci 数のリストを返しますが、 $f_n=start$ を含みませんが、 $f_n=stop$ は含みません。
find_root	閉区間 $[a, b]$ (または $[b, a]$) に於ける根を可能なら探す。ここで f は 1 変数の関数である。
floor	floor 関数
forall	$P(x)$ が S の全ての x で真であれば True と None を返します。もし、 S のある成分 x で P が True にならなければ、False と x を返します
forget	仮定を忘却する、あるいは無引数で全ての仮定を忘却する。ここで仮定は記号拘束の一種です
function	記号関数を生成します。
gaussian_binomial	ガウス分布を返します。
gcd	a と b の最大公約因子を返却
generic_power	m が非負であれば、 a の m 乗を返します。
get_memory_usage	記憶の状態を返します。
hex	整数、あるいは長整数の 16 進表現を返す
imag	x の虚部を返す
imaginary	複素数の虚部を返す
integer_ceil	x の四捨五入を返す。
integer_floor	整数で $\leq x$ を満たす最大の整数を返す
integral	対象 x の不定積分を返す
integrate	積分
interval	a と b の間にある整数(a と b は整数)
is_AlgebraElement	x の型が AlgebraElement であれば True を返す
is_commutative	
is_ComplexNumber	
is_even	偶数、つまり、2 で割り切れるかどうかを返します。
is_Functor	
is_Infinite	

is_Integer	
is_odd	奇数であるかどうかを返す。定義による is_even の補完
is_power_of_two	2 の冪乗のときだけ True を返す。
is_prime	素数であれば True を返し、そうでなければ False を返す。
is_prime_power	素数の冪乗であれば True、それ以外は False。
is_pseudoprime	擬素数であれば True、そうでなければ False。
is_RealNumber	RealNumber 型であれば True を返し、 meaning that it is an element of the MPFR real field with some precision.
is_Set	SAGE Set であれば True を返す
is_square	平方数であるかどうかを返します。平方数であれば平方根を返し、そうでなければ None を返します。
is_SymbolicExpression	
isqrt	整数の平方根を返します。つまり、平方根を越えない整数を返します。
laplace	Laplace 変換を計算して返します。
latex	SAGE の対象を latex(...) で latex の書式に変換します。
lcm	a と b の最小公倍数、The least common multiple of a and b, or if a is a list and b is omitted the least common multiple of all elements of v.
len	列や写像の成分の総数を返します。R
lim	与えられた方向からの変数 v の a への極限を返す
limit	与えられた方向からの変数 v の a への極限を返す
list	list() -> new list, list(sequence) -> new list で sequence の成分で初期化されます。
list_plot	list_plot は単与件リストを取り、これはタプル(i, di)のリストとなります。ここで i は 0 から len(data)-1 迄で、di は与件の i 番目の値です。そして、点はこれらのタプル上に配置されます。list_plot はまたタプル (dxi, dyi) のリストを取ります。ここで dxi は X 座標を表現する i 番目の与件、dyi は i 番目の値で、plotjoined=True なら全ての点が繋がれて描かれます。
load	名前が filename のファイルから SAGE の対象を読み込みます。ファイル名には .sobj という修飾子がかもしなければ追加されます。註: load と呼ばれる特殊な SAGE 命令(Python では使えません)もあり、load filename.sage と入力して用います。
loads	s=dumps(x) を使うことで文字列 s に吐き出された対象 x を復元します。
log	底を 2 とする自然対数。
matrix	行列を生成

max	単列の変数で、その最も大きな元を返します。二つ、あるいはそれ以上の引数に対しては、最大の引数を返します。
min	単列の変数で、その最も小さな元を返します。二つ、あるいはそれ以上の引数に対しては、最小の引数を返します。
minimal_polynomial	x で最小多項式を返す
mod	
mrange	与えられた大きさと型の複合リストを返します。
mul	リストの成分の積を返します。
next_prime	整数 n の次に大きな整数
next_prime_power	n が素数の時に、n よりも大きく、n に最も近い素数を返します
norm	ノルムを返す
normalvariate	正規分布
nth_prime	
number_of_arrangements	arrangements(mset, k) の大きさを返します。
number_of_combinations	combinations(mset, k) の大きさを返します。
number_of_derangements	derangements(mset) の大きさを返します。
number_of_divisors	因子の数を返します。
number_of_permutations	permutations(mset) の大きさを返します。
numerator	分子を返す
numerical_integral	区間 xmin から xmax で関数の数値積分を返します。R
numerical_sqrt	x の平方根を返します
oct	整数や多倍長整数の 8 進数表現を返します。
order	x の次数を返す。x が環、あるいは可換環の元であれば、これは x の additive order となる。
parametric_plot	parametric_plot はリストやタプルとしての二つの関数を取り、最初の関数が x 座標を与え、二番目の関数が y 座標を与えるグラフを描きます。
parent	定義されていれば x.parent()、そうでなければ type(x) を返します。
permutations	mset として丁度同じ成分を含み、順序が異なるリストによる置換の表現。
pg	置換群。SAGE では置換は分離した円環表示を使って置換を表現する文字列か、分離した円環で表現されたタプルのリストの何れかで表現されています。
pi	円周率
plot	
pow	二つの引数を取り、 x^y と同値になります。三引数では、 $(x^y) \% z$ と同値

	になります、もっと効率的かもしれません(e.g. long に対し)
power_mod	The m-th power of a modulo the integer n.
prange	start と stop-1 の間の素数のリスト。
previous_prime	n よりも小さな素数で最大の物を返します。
previous_prime_power	n よりも小さな素数の冪乗で最大の物を返します。
prime_divisors	The prime divisors of the integer n, sorted in increasing order.
prime_factors	The prime divisors of the integer n, sorted in increasing order.
prime_powers	List of all positive primes powers between start and stop-1, inclusive.
primes	start から stop-1 の間の全ての素数の列を返します。 .
primes_first_n	最初の n 個の素数を返します。
prod	リスト x の元の積を返します。
quo	Return the quotient object x/y, e.g., a quotient of numbers or of a polynomial ring x by the ideal generated by y, etc.
quotient	Return the quotient object x/y, e.g., a quotient of numbers or of a polynomial ring x by the ideal generated by y, etc.
random	区間[0, 1]での乱数を返します。
random_prime	2 と n の間の素数 p をランダムに返します(つまり、 $2 \leq p \leq n$)。
randrange	範囲(start, stop[, step])からの乱数を選出します
range	Returns a list containing an arithmetic progression of integers.
rational_reconstruction	この関数は x/y を計算しようとします, ここで x/y は有理数です。
real	x の実部を返します。
reduce	Apply a function of two arguments cumulatively to the items of a sequence, from left to right, so as to reduce the sequence to a single value.
repr	Return the canonical string representation of the object.
reset	Delete all user defined variables, reset all globals variables back to their default state, and reset all interfaces to other computer algebra systems. If vars is specified, just restore the value of vars and leave all other variables alone (i.e., call restore).
restore	大域変数を再定義することで、それらの初期値に戻します。
round	Round a number to a given precision in decimal digits (default 0 digits). This always returns a real double field element.
sample	Chooses k unique random elements from a population sequence.
save	Save obj to the file with name filename, which will have an .sobj

	extension added if it doesn't have one. This will *replace* the contents of filename.
save_session	Save all variables that can be saved wto the given filename.
search	Return (True,i) where i is such that $v[i] == x$ if there is such an i, or (False,j) otherwise, where j is the position that a should be inserted so that v remains sorted.
search_doc	Full text search of the SAGE HTML documentation for lines containing s.
search_src	Search sage source code for lines containing s.
sec	正割函数
sech	双曲正割函数
seed	
seq	A mutable list of elements with a common guaranteed universe, which can be set immutable.
set	順序のない単一元で構成される集まりを生成する。
show	グラフィックス対象を表示する
show_default	Set the default for showing plots using the following commands: plot, parametric_plot, polar_plot, and list_plot.
shuffle	
sigma	Return the sum of the k-th powers of the divisors of n.
simplify	式を簡易化する
sin	正弦函数
sinh	双曲正弦函数
sleep	
slice	Create a slice object. This is used for extended slicing (e.g. $a[0:10:2]$).
slide	Use latex(...) to typeset a SAGE object. Use %slide instead to typeset slides.
solve	単一の方程式や方程式系を与えられた変数に対して代数的に解く。
sorted	
sqrt	平方根函数。これは記号平方根です。
square_free_part	Return the square free part of x, i.e., a divisor z such that $x = z y^2$, for a perfect square y^2 .
srange	Return list of numbers $\forall \text{code}\{a, a+\text{step}, \dots, a+k*\text{step}\}$, where $a+k*\text{step} < b$ and $a+(k+1)*\text{step} > b$. The type of the entries in the

	list are the type of the starting value.
str	対象の文字列表現を返します。
subfactorial	Subfactorial or rencontres numbers, or derangements: number of permutations of n elements with no fixed points.
sum	Returns the sum of a sequence of numbers (NOT strings) plus the value of parameter 'start'
super	Typically used to call a cooperative superclass method.
symbolic_expression	
sys	This module provides access to some objects used or maintained by the interpreter and to functions that interact strongly with the interpreter.
tan	正接関数
tanh	双曲正接関数
taylor	Expands self in a truncated Taylor or Laurent series in the variable v around the point a , containing terms through $(x - a)^n$.
transpose	
trial_division	Return the smallest prime divisor \leq bound of the positive integer n , or n if there is no such prime.
two_squares	Write the integer n as a sum of two integer squares if possible; otherwise raise a ValueError.
type	対象の型を返します
union	x と y の結合をリストとして返します
uniq	Return the sublist of all elements in the list x that is sorted and is such that the entries in the sublist are unique.
valuation	$p > 0$ を割り切る整数 m による p の冪を返します
var	記号変数を生成します。
vars	引数なしで locals() と同値です。引数があれば object._dict_ と同値です。
vector	与えられた成分を持つ R 上のベクトルを返します。
version	SAGE の版を返す。
view	対象の latex 表現を計算します。註: ノートブックモードではこの関数は単純に出力に png 画像を埋め込みます。
walltime	wall time を返却します
xgcd	$g = s*a + t*b = \gcd(a, b)$ を満たす三個の整数 (g, s, t) を返します。
xinterval	Iterator over the integers between a and b , inclusive.
xrange	range() に似ていますが、リストを返す代わりに要求された範囲の数を生成

	する対象を返します。
zip	Return a list of tuples, where each tuple contains the i-th element from each of the argument sequences.

表 3: SSAGE 函数の一部

967 3.22 SAGE 函数の情報を得る

968 表 3 には各函数がどのような事をするか短い解説の一覧があります。しかしながら、これらの
 969 函数をどのように使うかを示す情報としては不十分です。任意の函数の追加の情報を得る一
 970 つの方法は疑問符 '?' をワークシートのセルの中で、函数名のうしろに続けて入力してから
 971 <tab>キーを押すことです：

```

972 is_even?<tab>
973 |
974 File: /opt/sage-2.7.1-debian-32bit-i686-
975 Linux/local/lib/python2.5/site-packages/sage/misc/functional.py
976 Type: <type 'function'>
977 Definition: is_even(x)
978 Docstring:
979     Return whether or not an integer x is even, e.g., divisible by 2.
980     EXAMPLES:
981     sage: is_even(-1)
982     False
983     sage: is_even(4)
984     True
985     sage: is_even(-2)
986     True
  
```

987 表示される灰色のウィンドウには函数についての次の情報が含まれています：

988 **File:** 函数を実装するソースコードを含むファイル名を与えます。函数がどのように実装さ
 989 れているかを見る為、あるいは編集する為にファイルを探そうとしていれば便利です。

990 **Type:** 参照している情報サービスに引き渡された名前の対象の型を示します。

991 **Definition:** 函数の呼び出され方を示します。

992 **Docstring:** 函数のソースコードの中に置かれた文書文字列を表示します。

993 表3で挙げられた任意の関数のヘルプ、あるいはSAGEのマニュアルがこの方法を使って得ら
994 れます。また、二つの疑問符'??'を関数名のうしろに置いて、<tab>キーを押せば、関数の
995 ソースコードが表示されます。

996 3.23 利用者が入力した関数についても情報が使える

997 情報サービスは利用者入力の関数についての情報を得ることに使え、情報サービスがどの
998 ように動作するかをより良く理解すれば、一度、これを入力することでご利益があります。
999 もしも、貴方が既に現時点でのワークシート上でそれを実行していたのであれば、再び
1000 addnums 関数を入力して、それを実行してください:

```
1001 def addnums(num1, num2):  
1002     """  
1003     num1 と num2 の和を返す。  
1004     """  
1005     answer = num1 + num2  
1006     return answer
```

```
1007 #関数を呼出し、3に2を加える。  
1008 a = addnums(2, 3)  
1009 print a  
1010 |  
1011 5
```

1012 そして、ここで新しく再入力した関数についての情報が前節の手法を使って得られます:

```
1013 addnums?<tab>  
1014 |  
1015 File: /home/sage/sage_notebook/worksheets/root/9/code/8.py  
1016 Type: <type 'function'>  
1017 Definition: addnums(num1, num2)  
1018 Docstring:  
1019 Returns the sum of num1 and num2.
```

1020 これは関数に関して表示された情報が関数のソースコードから得られた事を示しています。

1021 3.24 SAGE に含まれる関数を用いた例

1022 次の短いプログラムは 表3 に挙げられていた幾つかの関数がどのように用いられるかを示
1023 すものです:

1024

```
1025 #1 から 10 までの数の総和。
1026 add([1,2,3,4,5,6,7,8,9,10])
1027 |
1028     55

1029 #1 ラジアン余弦関数の値
1030 cos(1.0)
1031 |
1032     0.540302305868140

1033 #15/64 の分母
1034 denominator(15/64)
1035 |
1036     64

1037 #20 の全ての約数のリストを得る
1038 divisors(20)
1039 |
1040     [1, 2, 4, 5, 10, 20]

1041 #40 と 132 の最大公約数を求める。
1042 gcd(40,132)
1043 |
1044     4

1045 #2, 3, と 4 の積。
1046 mul([2,3,4])
1047 |
1048     24

1049 #リストの長さ。
1050 a = [1,2,3,4,5,6,7]
1051 len(a)
1052 |
1053     7

1054 #0 から 10 までの整数リストを生成。
1055 a = srange(11)
1056 a
1057 |
1058     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10]

1059 #0.0 と 10.5 の間の実数を 0.5 刻みでリストで出力。
1060 a = srange(11,step=.5)
```



```
1061 a
1062 |
1063     [0.000000, 0.500000, 1.000000, 1.500000, 2.000000, 2.500000,
1064     3.000000, 3.500000, 4.000000, 4.500000, 5.000000, 5.500000,
1065     6.000000, 6.500000, 7.000000, 7.500000, 8.000000, 8.500000,
1066     9.000000, 9.500000, 10.00000, 10.50000]
```

```
1067 #-5から5までの整数を含むリストを生成。
1068 a = srange(-5,6)
1069 a
1070 |
1071     [-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5]
```

```
1072 #The zip() function takes multiple sequences and groups
1073 #parallel members inside tuples in an output list. One
1074 #application this is useful for is creating points from
1075 #table data so they can be plotted.
```

```
1076 a = [1,2,3,4,5]
1077 b = [6,7,8,9,10]
1078 c = zip(a,b)
1079 c
1080 |
1081     [(1, 6), (2, 7), (3, 8), (4, 9), (5, 10)]
```

1082 3.25 for 文で srange() と zip() を利用

1083 for 文用に手動で列を構成する代わりに、srange()が自律的に列を生成することに使えま
1084 す：

```
1085 for t in srange(6):
1086     print t,
1087 |
1088     0 1 2 3 4 5
```

1089 for 文はまた zip() 関数の利用から複数の列を並べて使うことができます：

```
1090 t1 = (0,1,2,3,4)
1091 t2 = (5,6,7,8,9)
1092 for (a,b) in zip(t1,t2):
```

```
1093     print a,b
1094 |
1095     0 5
1096     1 6
1097     2 7
1098     3 8
1099     4 9
```

1100 3.26 リスト内包(list comprehension)

1101 ここまでで、我々は if 文、for 文、list と **関数** が、個別に、そして互いに用いられた時
1102 に、それぞれ非常に強力なことを見えています。より強力で、リスト内包(list
1103 comprehension)と呼ばれる特殊な文では、最小の構文でこれらを合わせて使うことが
1104 できます。

1105 ここでリスト内包の簡略化した構文は：

```
1106 [ 式 for 変数 in 列 [if 条件] ]
```

1107 リスト内包が行うことは、列を通じて、各列の成分をかわるがわる指定した変数で置いた反
1108 復処理を行うことです。その式はまた変数を含み、変数の中に数を置くのにしたがって式が
1109 評価され、その結果は新しいリストに置かれます。列の中の全ての数が処理されると、新し
1110 いリストが返されます。

1111 次の例では、t が変数で、2*t が式です。そして [1, 2, 3, 4, 5] が列です：

```
1112 a = [2*t for t in [0,1,2,3,4,5]]
1113 a
1114 |
1115     [0, 2, 4, 6, 8, 10]
```

1116 列を手で生成する代わりに、列の自律的生成で xrange() 関数がしばしば利用さ：

```
1117 a = [2*t for t in xrange(6)]
1118 a
1119 |
1120     [0, 2, 4, 6, 8, 10]
```

1121 おまけの if 文は、リスト内包で新しいリストで置き換えられる結果を篩にかける為に用い
1122 ることができます：

```
1123 a = [b^2 for b in range(20) if b % 2 == 0]
```

```
1124 a
```

```
1125 |
```

```
1126 [0, 4, 16, 36, 64, 100, 144, 196, 256, 324]
```

1127 この場合、2で割り切れるものに対する結果のみが出力リストに置かれます。

1128 4 オブジェクト指向プログラミング

1129 この章の目的は、オブジェクト指向の SAGE プログラムがどのように動作し、どの様に問題を
1130 解く為の用いられるかといった背後にある**主要な概念を紹介する**事です。貴方が僅か、ある
1131 いは全くオブジェクト指向プログラミング (OOP) の経験がないと仮定しており、貴方が OOP
1132 を十分理解することで問題を解く為に SAGE の対象を効果的に使えるようになる事を目的にし
1133 ています。もしも、この OOP の詰込が完全に飲み込めなかったとしても心配する必要はあり
1134 ません。何故なら、貴方自身、何もない状態から対象をプログラムすることに必要とされる
1135 技能を持っていなくても、問題を解く為に SAGE の対象が使えるからです。そういいながら
1136 も、この章では、どの様に真っ新たな状態から対象をプログラムするかを示すことで、貴方が
1137 SAGE の組込の対象がどの様に動作するかがより良く理解出来るでしょう。

1138 4.1 オブジェクト指向な心の書換

1139 私の意見では、この類のプログラミングで貴方が遂行する物事で最も難しい物の一つが手続
1140 き型プログラミングの世界からオブジェクト指向プログラミングの世界へと心の切り替えを
1141 行うことです。問題はオブジェクト指向プログラミングが手続き型プログラミングよりも必
1142 然的に難しいことではありません。問題はプログラミングの問題を解くその手法に違いが大
1143 きくて、貴方が本当に”それを会得する”前に幾つかの心的な書換が生じなければならないこ
1144 とです。この心理的な書換は貴方がオブジェクト指向プログラムを書いたり、一体全体何が
1145 OOP であるかを本当に理解する為に苦勞してオブジェクト指向の書籍に耽溺するに従って、
1146 非常にゆっくりと生じる過程です。

1147 まず初めに、貴方が何か特別で強力な何かが進行していると思うでしょうが、それに固く結
1148 びついた貴方の労力を逃してしまうでしょう。貴方がやっと”理解する”事は、普通、明る
1149 い光が差し込むように一挙に生じる事ではありません。それは薄明にとても似たもので、貴
1150 方の意識の背後で、その耀きをゆっくりと感じられるものです。貴方が関わる新しいプログ
1151 ラミングの問題の各々に対し、貴方の心の正面側では依然、それを解く為に手続的な計画を
1152 作成しようとするでしょう。しかし、貴方の心の背後で、この薄明がオブジェクト指向的な
1153 戦略(最初はぼんやりとしていますが、やがて明瞭さを増してゆきます)を表現し、それでま
1154 た問題が解けることに気付くでしょう。そして、これらのオブジェクト指向的な戦略がやが
1155 て興味深い物となって、やがて、それらにより多くの時間を費やしている自分に気付く事
1156 なるでしょう多くのプログラミングの問題が、それらを解く為に豊かなオブジェクト指向的
1157 な戦略と云う生産の引き金を引くことになったその時に、輝くオブジェクト指向的な部分が
1158 貴方の心に訪れるでしょう。

1159 4.2 属性と振舞

1160 オブジェクト指向プログラミングはソフトウェア設計の哲学であり、そこでソフトウェアは
1161 物理的な世界で対象が作用する方法に似た動作をさせます。全ての物理的な対象は**属性**
1162 (**attribute**)と**振舞**(**behavior**)があります。一つの例として、典型的な事務所の椅子に
1163 は、色、コロの数、そして物質的な型を属性とし、スピン、ロールや高さの設定が振舞
1164 です。

1165

1166 ソフトウェア対象は物理的対象の様に働かせられ、それらはまた属性と振舞を持っていま
1167 す。ソフトウェア対象の**属性**は**インスタンス変数**と呼ばれる特殊な変数で保持され、その
1168 **振舞**は**メソッド**(これは**構成員関数**(**member function**)とも呼ばれます)で保持されるプ
1169 ログラムで決定されます。メソッドは標準の関数と似ていますが、ただ、それは”自由に
1170 漂っている”代わりに対象に関連付けられています。SAGE では、インスタンス変数とメソッ
1171 ドはしばしば**属性**として参照されます。

1172 対象が生成されると、メソッドの呼出し、あるいは起動を意味するメッセージの送付に使わ
1173 れます。

1174 椅子の場合は、`char.spin(3)`メッセージでその椅子を 3 回スピンさせる事を伝え、さらに
1175 は、`chair.setHeight(32)`メッセージでその椅子を 32 センチメートルに設定する事を伝える
1176 ことを想像してもよいでしょう。

1177 4.3 類 (対象を生成する為の設計図)

1178 **類**(**class**)は対象を構築する為に用いられる**設計図**として考えられ、概念的に家の設計図に
1179 似たものです。建築家は、与えられた家がどのように構築されるべきか、どの様な材料が使
1180 われ、様々な寸法がどうあるべきか等が正確に定義された設計図を使います。設計図が終わ
1181 ると、ひとつ、あるいはそれ以上の家の建設でそれが使えますが、何故なら、設計図には家
1182 をどの様に建築するかと言ったことが記述された情報を含んでいるからですが、家それ自体
1183 にはありません。プログラマーが**類**を生成することは建築家が**家の設計図**を作成するのと非
1184 常に似ていますが、建築家が図面台や CAD システムを使って設計図を作るのに対し、プログ
1185 ラマーはテキストエディタや IDE(総合開発環境: Integrated Development Environment)を
1186 使って類を開発する点が異なります。

1187

1188 4.4 オブジェクト指向プログラムは必要に応じて対象の生成と破壊を 1189 行う

1190 次の類比はオブジェクト指向プログラム内部でソフトウェア対象が必要に応じてどのように
1191 生成され、破壊されるかを解説するものです。対象の生成はまたインスタンスによる表現と

1192 も呼ばれますが、それは対象を定義する類(設計図)が対象インスタンスを生成するために用
1193 いられているからです。対象を破壊し、記憶と他の資源を再要求してそれを用いる行為のこと
1194 を**ゴミ収集(garbage collection)**と呼びます。

1195 オブジェクト指向プログラムに似せた手法で与えられたジェット旅客機を操作し、それか
1196 ら、そのジェット機が大西洋を渡ってニューヨークからロンドンに飛ぶ準備をしているところ
1197 を想像してください。離陸の前に、飛行機の全ての部品の設計図が滑走路に運び込まれ、
1198 飛行機の構築に必要とされる部品の全てを非常に素早く製作する為に一団の作業員が送り込
1199 まれます。各部品が出来上がると、飛行機の正しい位置に据え付けられ、短い時間の間に飛
1200 行機が完成して利用できるようになります。乗客はジェット機に乗り込み、そしていよいよ
1201 離陸です。

1202 飛行機が地面を離れると滑走車輪を解体してしまいます(ゴミ収集されます)。なぜなら、そ
1203 れらは飛行中は不要で、大西洋を渡っている間に車輪を出して風切り音をたてていることは
1204 全く燃料の無駄だからです。だからといって心配する必要はありません。と言うのもロンド
1205 ンに着陸する前に滑走車輪を適正な設計図(類)を使って再構成してしまえばよいからです。

1206 離陸後の数分後に、パイロットは飛行機のジェットエンジンを作った会社が丁度、新モデル
1207 を出し、今使っているものよりも 15%以上燃料効率がよいので飛行中でも航空会社が飛行機
1208 のエンジンの取替えを行うとの連絡を受け取ります。航空会社はニューヨークから新しいエ
1209 ンジンの設計図を飛行機に送り込み、これらは新しい三個のエンジンの組立て(インスタンス
1210 化)で用いられます。新しいエンジンが組み立てられると、古い三個のエンジンは同時に停止
1211 させられて、新しいエンジンで置き換えられて解体されてしまいます。エンジンの取替えは
1212 素早く行われ、旅客はこの取替えが行われたことに気付きさえしません。この飛行には世界
1213 的重要人物が乗っており、途上で遭遇した敵対機が我らがパイロットに航路を外れるように
1214 要求します。この要求に応じる代わりに、パイロットは設計図のライブラリから一群の 50mm
1215 機銃砲塔の設計図を取り出し、4 個の砲塔を構築し、それから飛行機の頭、腹、鼻と尻尾に
1216 それらを取り付けます。機銃からの多少の砲火は敵機を驚かせるに十分で、さっさと逃げて
1217 しまって、やがてレーダーからも消えてしまいました。残りの飛行は何も問題はありませ
1218 ん。飛行機がロンドンに近づくに従って機銃の砲塔は解体され、新しい着陸車輪が設計図を
1219 使って構成されると飛行機は無事に着陸します。旅客がターミナルに入ると、飛行機全体は
1220 解体されてしまいます。

1221 4.5 オブジェクト指向プログラム例

1222 次の二節では、**Hellos** という名前の単純なオブジェクト指向プログラムの話をします。最
1223 初の節ではプログラムを含んでいますが、そのプログラム自体を見やすいように何らの注釈
1224 も入れていません。第二の節では、注釈をしっかりと入れたプログラムがあり、そのプログラ

1225 ムがどのようにして動作するのが詳細に記述されています。

4.5.1 Hellos オブジェクト指向プログラム例 (無注釈)

```
1226 class Hellos:
1227     def __init__(self, mess):
1228         self.message = mess
1229
1229     def print_message(self):
1230         print"The message is: ", self.message
1231
1232     def say_goodbye(self):
1233         print "Goodbye!"
1234
1235     def print_hellos(self, total):
1236         count = 1
1237         while count <= total:
1238             print"Hello ", count
1239             count = count + 1
1240
1240         print " "
1241
1241     obj1 = Hellos("Are you having fun yet?")
1242     obj2 = Hellos("Yes I am!")
1243
1244     obj1.print_message()
1245     obj2.print_message()
1246     print " "
1247
1247     obj1.print_hellos(3)
1248     obj2.print_hellos(5)
1249
1249     obj1.say_goodbye()
1250     obj2.say_goodbye()
1251
1251     |
1252     The message is: Are you having fun yet?
1253     The message is: Yes I am!
1254
1255     Hello 1
1256     Hello 2
1257     Hello 3
1258
1259     Hello 1
1260     Hello 2
```

```
1261     Hello  3
1262     Hello  4
1263     Hello  5
1264
1265     Goodbye!
1266     Goodbye!
```

1267 4.5.2 Hellos オブジェクト指向プログラム例 (注釈付き)

1268 では、Hellos プログラムの詳細を見ることにしましょう。このプログラムの版は注釈を付
1269 けたものです。プログラム左側の行数とコロンはプログラムそれ自身ではなく、それらはプ
1270 ログラムの異なった部分を参照し易くする為に追加したものです。

```
1271 1:class Hellos:
1272 2:     """
1273 3:     Hellos は'類'であり、類は対象生成の青写真になります。
1274 4:     類はインスタンス変数(属性, Attribute) とメソッド(振舞, Behavior)
1275 5:     で構成されます。
1276 6:     """
1277 7:
1278 8:     def __init__(self, mess):
1279 9:         """
1280 10:         __init__ はコンストラクタが呼び出した組込の特殊な
1281 11:         メソッドです。コンストラクタ メソッドは、対象が生成
1282 12:         された時に一度だけ呼び出され、その作業は対象の構築を
1283 13:         完遂する事です。その対象が生成されたのちには、
1284 14:         そのコンストラクタはもはや用いられません。
1285 15:         コンストラクタの目的は'伝言'と呼ばれるインスタンス
1286 16:         変数を生成することで、それから、文字列を使って、
1287 17:         それを初期化します。
1288 18:         """
1289 19:
1290 20:         """
1291 21:         このコードはインスタンス変数を生成します。 この'設計図'
1292 22:         類から生成された対象インスタンスは任意のインスタンス変数
1293 23:         のそれ自身のユニークな複製を持っています。インスタンス
1294 24:         変数は対象の属性(あるいは状態)を保持します。
1295 25:         ここでの self 変数は現行の対象への参照を保ちます。
1296 26:
```



```
1297 27:         """
1298 28:         self.message = mess;
1299 29:
1300 30:
1301 31:
1302 32:     def print_message(self):
1303 33:         """
1304 34:         print_message はインスタンスメソッドであり、この類を使って、
1305 35:         生成された対象に '伝言を印字する' 振舞を与えます。
1306 36:         """
1307 37:         print"The message is: ", self.message
1308 38:
1309 39:
1310 40:
1311 41:     def say_goodbye(self):
1312 42:         """
1313 43:         say_goodbye はインスタンスメソッドであり、この類を使って、
1314 44:         生成された対象に 'goodby という' 振舞を与えます。
1315 45:         """
1316 46:         print "Goodbye!"
1317 47:
1318 48:
1319 49:
1320 50:     def print_hellos(self, total):
1321 51:         """
1322 52:         print_hellos はインスタンスメソッドで、Hello の数を
1323 53:         印字の引数として取り、画面にこの Hello を沢山印字し
1324 54:         ます。
1325 55:         """
1326 56:         count = 1
1327 57:         while count <= total:
1328 58:             print"Hello ", count
1329 59:             count = count + 1
1330 60:
1331 61:         print " "
1332 62:
1333 63:
1334 64: """
```

1335 65: 次のプログラムは二つの分離した Hellos 対象(インスタンス)

1336 66: を生成し、これらは変数 obj1 と obj2 としてそれぞれが表現

1337 67: されます。インスタント化された時に、ユニークな文字列

1338 68: 助変数が各対象に、引き渡され、この文字列は対象の状態を

1339 69: 初期化する為に用いられます。

```
1340 70:
1341 71: 対象が生成されると、それらに振舞を実行させる為に、
1342 72: メソッドを呼び出すことで対象に対して伝言が送られます。
1343 73: これは'対象の取り上げ'による参照(obj1 とします)で実行
1344 74: されますが、その参照ではうしろに点を置いて、呼び出したい。
1345 75: 対象の名前を記入します。
1346 76: ""
1347 77:
1348 78:obj1 = Hellos("Are you having fun yet?")
1349 79:obj2 = Hellos("Yes I am!")
1350 80:
1351 81:obj1.print_message()
1352 82:obj2.print_message()
1353 83:print " "
1354 84:
1355 85:obj1.print_hellos(3)
1356 86:obj2.print_hellos(5)
1357 87:
1358 88:obj1.say_goodbye()
1359 89:obj2.say_goodbye()
```

1360 1行での Hellos 類は **class** 文を使って定義され、類名は習慣から大文字で始めます。類名が
1361 複数の語で構成されていれば、各語の最初の文字を大文字にして、残りの全ての文字を小文
1362 字で記述します(例えば、HelloWorld)。その類は行1で始まり行61で終わります。この行61
1363 はインデントをつけたコードの最終行です。類の一部としての全てのメソッドとインスタ
1364 **ンス変数**は類のインデント付けられた区画の中にある必要があります。

1365 Hellos 類は一つの**コンストラクタ**メソッドを8行目に、28行目で生成された一つの**インス**
1366 **タンス変数**と32行目、41行目と50行目にそれぞれ3個の**インスタンスメソッド**を含んで
1367 います。**インスタンス変数**の目的は対象に、与えられた類から生成された他の対象と異な
1368 るユニークな**属性**を与えることです。**インスタンスメソッド**の目的は各対象にその**振舞**を
1369 与えることです。対象の全てのメソッドは、対象の**インスタンス変数**に接続し、これらの**イ**
1370 **ンスタンス変数**はこれらのメソッドの中のプログラムによって接続が可能です。**インスタン**
1371 **ス変数名**は**関数名**と同様の習慣に従います。

1372 8行目のメソッドは**コンストラクタ**と呼ばれる特殊なメソッドです。 **コンストラクタ**
1373 **メソッド**は対象が生成される時に一度だけ呼び出され、その目的は対象の構築を完遂するこ
1374 とです。対象が生成されたあとには、その**コンストラクタ**は最早使われません。8行目の**コ**
1375 **ンストラクタ**の目的は、Hellos 対象の **message** **インスタンス変数**を **Hello** 型の新しい対象
1376 が生成されたとき(78行と79行を参照)にそれに引き渡される文字列で初期化する事です。

1377 全てのインスタンスメソッドは呼び出されたメソッドからの特定の対象に対する参照を含む
1378 一つの引数を持ちます。この引数は常に左端の位置に置かれ、通常、この位置に置かれる変
1379 数を **self** と呼びます。この **self** 変数は特定の対象のインスタンス変数を生成したり、利用
1380 する為に使われます。

1381 28 行目の **self.message=mess** でコンストラクタ **mess** 変数に引き渡された対象を取
1382 り、**message** と呼ばれるインスタンス変数に引き渡します。インスタンス変数は通常の変数
1383 であるかの様に、割り当てによって生成されます。この点演算子 **'.'** は対象のインスタンス変
1384 数に対象を参照する変数とインスタンス変数名の間に (**self.message** や **obj1.message** の様
1385 に)置かれることで対象のインスタンス変数を参照する為に用いられます。

1386 32 行目, 41 行目と 50 行目のメソッドは **Hellos** 類を用いて生成された対象に、それらの振舞
1387 を与えます。 **print_message()** はその対象の **message** インスタンス変数にある文字列を印字
1388 するという振舞を実行し、 **say_goodbye()** メソッドは、文字列 **"Goodbye!"** を印字するという
1389 振舞を実行します。 **print_hellos()** メソッドは引数として一つの整数を取り、その回数
1390 程、 **'Hello'** という言葉を印字します。メソッドの名付け方は函数名で用いたものと同様で
1391 す。

1392 **Hellos** 類の下のコードは二つの分離した対象(インスタンス)を生成し、これらは変数 **obj1**
1393 と変数 **obj2** にそれぞれ割り当てられます。対象はその類名を括弧に続けて入力することで
1394 生成されます。括弧の中に置かれた引数はコンストラクタメソッドに引き渡されます。
1395 **Hellos** 類が呼出されたとき、文字列がそのコンストラクタメソッドに引き渡され、この文
1396 字列は対象の**状態**の初期化で用いられます。対象の状態はそのインスタンス変数の内容で決
1397 定されます。もし、対象のインスタンス変数が変更されると、対象状態も変更されま
1398 す。**Hellos** 対象は **message** と呼ばれるインスタンス変数のみを持っているので、それらの
1399 状態はこの変数によって決定されます。

1400 対象が生成されると、それらの振舞はメソッドの呼出によって要求されます。
1401 これは参照する変数(**obj1** と呼びましょう)による**"対象の摘み上げ"**によって実行され、
1402 点のうしろにこの変数を置き、呼び出したい対象のメソッドの内の一つの名前を入力し、
1403 その引数のうしろに括弧を続けます。

1404 4.6 SAGE 類(Class)と対象(Object)

1405 SAGE の函数は沢山の機能を持っているので、SAGE の機能の殆どは**類**と**類**から実体化された
1406 **対象**に含まれます。SAGE の類と対象は説明するには莫大な情報を表現します。とはいえ、
1407 貴方が可能な限り早く SAGE の対象を使って作業を始めることが出来るように、より簡単な素

1408 材を最初に示すことにしましょう。

1409 4.7 SAGE の対象について情報を得ること

1410 セルに次のコードを入力し、それを実行てみましょう：

```
1411 x = 5
1412 print type(x)
1413 |
1414     <type 'sage.rings.integer.Integer'>
```

1415 我々は既に type() 関数を整数の型を定める為に用いていますが、ここで我々は型がどのよう
1416 なものであるか詳細を説明する事が出来ます。Sage.rings.integer.Integer と疑問符 '?' に
1417 続けて入力して <tab> キーを押してみましよう：

```
1418 sage.rings.integer.Integer?<tab>
1419 |
1420 File:/opt/sage-2.7.1-debian-32bit-i686-
1421 Linux/local/lib/python2.5/site-packages/sage/rings/integer.so
1422 Type:     <type 'sage.rings.integer.Integer'>
1423 Definition: sage.rings.integer.Integer([noargspec])
1424 Docstring:
1425
1426     The class{Integer} class represents arbitrary precision
1427     integers. It derives from the class{Element} class, so
1428     integers can be used as ring elements anywhere in SAGE.
1429
1430     begin{notice}
1431     The class class{Integer} is implemented in Pyrex,
1432     as a wrapper of the GMP mpz_t integer type.
1433     end{notice}
```

1432 この情報は、sage.rings.integer.Integer が本当に類であり、整数類を生成できる事を示し
1433 ています。また、貴方が二つの疑問符'??'を類名のうしろに置いて、<tab>キーを押せば、類
1434 のソースコードが表示されます。

1435 では、別のセルで x. と入力し、それから <tab> キーを押しましょう：

```
1436 x.<tab>
1437 |
```

1438	x.additive_order	x.gcd	x.numerator
1439	x.base_base_extend	x.inverse_mod	x.ord
1440	x.inverse_of_unit	x.order	x.parent
1441	x.base_extend	x.is_nilpotent	x.plot
1442	x.base_extend_canonical	x.is_one	x.powermodm_ui
1443	x.is_perfect_power	x.powermod	x.quo_rem
1444	x.base_extend_recursive	x.is_power	x.rename
1445	x.base_ring	x.is_power_of	x.reset_name
1446	x.binary	x.is_prime	x.save
1447	x.category	x.is_prime_power	x.set_si
1448	x.ceil	x.is_pseudoprime	x.set_str
1449	x.coprime_integers	x.is_square	x.sqrt
1450	x.crt	x.is_squarefree	x.sqrt_approx
1451	x.db	x.is_unit	x.square_free_part
1452	x.degree	x.is_zero	x.str
1453	x.denominator	x.isqrt	x.substitute
1454	x.digits	x.jacobi	x.test_bit
1455	x.div	x.kronecker	x.val_unit
1456	x.lcm	x.subs	x.valuation
1457	x.divides	x.leading_coefficient	x.version
1458	x.dump	x.list	x.xgcd
1459	x.dumps	x.mod	x.parent
1460	x.exact_log	x.multiplicative_order	x.plot
1461	x.factor	x.next_prime	x.rename
1462	x.factorial	x.next_probable_prime	x.reset_name
1463	x.floor	x.nth_root	x.powermodm_ui

1464 表示させる灰色のウィンドウには対象が包含する全てのメソッドが含まれています。これら
 1465 の任意のメソッドがマウスで選択されると、その名前がドット演算子のうしろのセルに置か
 1466 れます。ここでは、`is_prime` メソッドを選択しましょう。その名前がセルに置かれた時点
 1467 で、疑問符'?'をそのうしろに入力して、<tab>キーを押すと、このメソッドの情報が得られ
 1468 ます:

```

1469 x.is_prime?
1470 |
1471 File:      /opt/sage-2.7.1-debian-32bit-i686-
1472 Linux/local/lib/python/site-packages/sage/rings/integer/pyx
1473 Type:      <type 'builtin_function_or_method '>
1474 Definition: x.is_prime()

1475 Docstring:

1476         Returns True if self is prime

1477         EXAMPLES:

```

```
1478 sage: z = 2^31 - 1
1479 sage: z.is_prime()
1480 True
1481 sage: z = 2^31
1482 sage: z.is_prime()
1483 False
```

1484 定義の節から `is_prime()` メソッドが引数を引き渡されるに呼び出し事が分かり、Docstring
1485 の節から、このメソッドが対象が素数であれば True を返すものであることが分かります。次
1486 のコードは変数 `x`(これは 5 を包含しています)が `is_prime()` メソッドを呼び出す為に用いら
1487 れています:

```
1488 x.is_prime()
1489 |
1490 True
```

1491 4.8 対象のメソッドのリスト

1492 リストは対象であり、それ故、リストには便利な機能を持ったメソッドを含んでいます:

```
1493 a = []
1494 a.<tab>
1495 |
1496 a.append    a.extend    a.insert    a.remove    a.sort
1497 a.count     a.index     a.pop       a.reverse
```

1498 次のプログラムは対象としてのリストの幾つかのメソッドのデモです:

```
1499 # リストの末端に対象を追加.
1500 a = [1,2,3,4,5,6]
1501 print a
1502 a.append(7)
1503 print a
1504 |
1505 [1, 2, 3, 4, 5, 6]
1506 [1, 2, 3, 4, 5, 6, 7]
```

```
1507 # リストに対象を挿入.
1508 a = [1,2,4,5]
1509 print a
1510 a.insert(2,3)
1511 print a
1512 |
```

```
1513 [1, 2, 4, 5]
1514 [1, 2, 3, 4, 5]

1515 # リストの成分を並び替え.
1516 a = [8,2,7,1,6,4]
1517 print a
1518 a.sort()
1519 print a
1520 |
1521 [8, 2, 7, 1, 6, 4]
1522 [1, 2, 4, 6, 7, 8]
```

1523 4.9 継承による類の拡張

1524 オブジェクト技術は微妙ですが強力です。これは複雑なものを扱う為の幾つかの機構をもつ
1525 ており、**類継承(class inheritance)**はそのうちの一つの物です。**類継承**は、最小の処
1526 理で、他の類(親の類、スパークラス、あるいはベースクラスと呼ばれます)のインスタンス
1527 変数やメソッドの全てを得たり、あるいは継承する為の類の能力です。親の類から継承され
1528 た類のことを**子供の類**、あるいは**サブクラス**と呼びます。この意味は、子供の類はその親
1529 で実行出来るものが利用可能だからです。

1530 次のプログラムでは、Person 類が組込みの object 類から継承し、ArmyPrivate 類が Person
1531 類から継承するという類継承を示します：

```
1532 class Person(object):
1533     def __init__(self):
1534         self.rank = "I am just a Person, I have no rank."
1535
1536     def __str__(self):
1537         return "str: " + self.rank
1538
1539     def __repr__(self):
1539         return "repr: " + self.rank

1540 class ArmyPrivate(Person):
1541     def __init__(self):
1542         self.rank = "ArmyPrivate."
```

```
1543 a = object()
1544 print type(a)
```

```
1545 b = Person()
1546 print type(b)

1547 c = ArmyPrivate()
1548 print type(c)
1549 |
1550     <type 'object'>
1551     <class '__main__.Person'>
1552     <class '__main__.ArmyPrivate'>
```

1553 類が生成されたあとに、このプログラムは **object** 型の対象を変数 'a' に割当て、**Person** 型
1554 の対象を変数 'b' に割当て、**ArmyPrivate** 型の対象を変数 'c' に割当ててインスタンス化し
1555 ます。

1556 次の処理は任意の対象の継承階層を表示させることに使えます。上のプログラムを実行した
1557 あとで別のセルでそれを実行すれば、**ArmyPrivate** 類の継承階層が表示されます。
1558 (この処理がどの様にして動作するか考え込まないように。使ってみましょう。):

```
1559 #対象の継承階層を表示します。註： このプログラムが度のように動作するのか
1560 #思い悩む必要はありません。
1561 def class_hierarchy(cls, indent):
1562     print '.'*indent, cls
1563     for supercls in cls.__bases__:
1564         class_hierarchy(supercls, indent+1)
```

```
1565 def instance_hierarchy(inst):
1566     print 'Inheritance hierarchy of', inst
1567     class_hierarchy(inst.__class__, 3)
```

```
1568 z = ArmyPrivate()

1569 instance_hierarchy(z)
1570 |
1571     Inheritance hierarchy of str: ArmyPrivate
1572     ... <class '__main__.ArmyPrivate'>
1573     .... <class '__main__.Person'>
1574     ..... <type 'object'>
```

1575 `instance_hierarchy` 関数がそれに引き渡された任意の対象の継承階層を表示します。この場
1576 合、**ArmyPrivate** 類がインスタンス化されて `instance_hierarchy` 関数に引き渡され、それか
1577 ら対象の継承階層が表示されるのです。その階層で最上の類、つまり、**object** 類ですが、
1578 これが最後に表示され、**Person** は **object** から継承し、**ArmyPrivate** は **Person** から継承
1579 します。

1580 4.10 対象類、dir() 関数、組込メソッド

1581 対象類は SAGE に組込まれており、いくつかの便利なメソッドを持っています。これらのメ
 1582 ソッドは非常に便利なので、多くの SAGE の類がその `object` 類から 1)直接、あるいは 2)間
 1583 接的に継承しますが、それは `object` 類から継承している類からの継承によります。
 1584 さて、`object` 類に含まれるメソッドを調べることで、継承プログラムについて議論を始め
 1585 ることにしましょう。`dir()` 関数は全ての属性(インスタンス変数とメソッドの双方を意味し
 1586 ます)を一覧表示するので、`object` 型の対象がどのメソッドを含むかを見るためにそれが使
 1587 えます：

```
1588 dir(a)
1589 |
1590 | ['__class__', '__delattr__', '__doc__',
1591 |  '__getattribute__', '__hash__', '__init__', '__new__', '__reduce__',
1592 |  '__reduce_ex__', '__repr__', '__setattr__', '__str__']
```

1593 二重の `'_'` で開始し、それで終わる名前が SAGE の一部であり、下線によって、これらの名前
 1594 がプログラマーが定義した名前と衝突する事がない様にしています。Person 類は `object` 類
 1595 からのこれらの属性の全てを継承しますが、それらの内の幾つかだけしか使いません。メ
 1596 ソッドが親の類から継承されたとき、子供の類はメソッドの親の実装を使ったり、親のもの
 1597 とは異なった振舞をするように再定義することが出来ます。

1598 先に議論した様に、`__init__` メソッドはコンストラクタであり、それは類を使って生成され
 1599 た新しい類の完全な構成を助けるものです。Person 類は `__init__` メソッドを再定義し
 1600 て、`rank` と呼ばれるインスタンス変数を生成して文字列 "I am just a Person, I have no
 1601 rank" をそれに割り当てます。 `__repr__` と `__str__` メソッドもまた Person 類で再定義
 1602 されています。`__repr__` メソッドは、対象の一部である対象の文字列表現を返します：

```
1603 |
1604 | repr: I am just a Person, I have no rank.
1605 |
```

1606 `__str__` 関数はまた対象の一部でもある対象の文字列表現を返しますが、`print` の様な文に
 1607 引き渡された時だけです：

```
1608 print b
1609 |
1610 | str: I am just a Person, I have no rank.
```

1611 `__str__` メソッドは通常、`__repr__` メソッドよりもより利用者にやさしい文字列を返す為に用

1612 いられますが、この例では、とても似た文字列が返されます。

1613 4.11 Sage.rings.integer.Integer 類の継承階層

1614 次のプログラムは sage.rings.integer.Integer 類の継承階層を表示します：

```

1615 # 類の継承階層を表示します。 註：このプログラムがどの様にして動作するのかを
1616 # 理解しなくても構いません。使ってみましょう。
1617 def class_hierarchy(cls, indent):
1618     print '.'*indent, cls
1619     for supercls in cls.__bases__:
1620         class_hierarchy(supercls, indent+1)

1621 def instance_hierarchy(inst):
1622     print 'Inheritance hierarchy of', inst
1623     class_hierarchy(inst.__class__, 3)

1624 instance_hierarchy(1)
1625 |
1626 Inheritance hierarchy of 1
1627 ... <type 'sage.rings.integer.Integer'>
1628 .... <type 'sage.structure.element.EuclideanDomainElement'>
1629 ..... <type 'sage.structure.element.PrincipalIdealDomainElement'>
1630 ..... <type 'sage.structure.element.DedekindDomainElement'>
1631 ..... <type 'sage.structure.element.IntegralDomainElement'>
1632 ..... <type 'sage.structure.element.CommutativeRingElement'>
1633 ..... <type 'sage.structure.element.RingElement'>
1634 ..... <type 'sage.structure.element.ModuleElement'>
1635 ..... <type 'sage.structure.element.Element'>
1636 ..... <type 'sage.structure.sage_object.SAGEObject'>
1637 ..... <type 'object'>

```

1638 次の解説では、私は節約の為に類名の”sage.xxx.xxx...”の頭の部分を外す事にします。

1639 instance_hierarchy 函数からの出力は、数字 1 が類型 Integer であることを示していま
1640 す。その出力は Integer は EuclideanDomainElement から継承

1641 し、EuclideanDomainElement は PrincipalIdealDomainElement から継承し..等であ
1642 ることを示しています。階層の頂点(この一覧の底になります)には SAGEObject が object か
1643 ら継承しています。

1644 ここで二つの他の広く使われる SAGE 類の継承階層は：

```

1645 instancehierarchy(1/2)
1646 |
1647 Inheritance hierarchy of 1/2

```

```
1648 ... <type 'sage.rings.rational.Rational'>
1649 .... <type 'sage.structure.element.FieldElement'>
1650 ..... <type 'sage.structure.element.CommutativeRingElement'>
1651 ..... <type 'sage.structure.element.RingElement'>
1652 ..... <type 'sage.structure.element.ModuleElement'>
1653 ..... <type 'sage.structure.element.Element'>
1654 ..... <type 'sage.structure.sage_object.SAGEObject'>
1655 ..... <type 'object'>
```

```
1656 instancehierarchy(1.2)
```

```
1657 |
1658 Inheritance hierarchy of 1.200000000000000
1659 ... <type 'sage.rings.real_mpfr.RealNumber'>
1660 .... <type 'sage.structure.element.RingElement'>
1661 ..... <type 'sage.structure.element.ModuleElement'>
1662 ..... <type 'sage.structure.element.Element'>
1663 ..... <type 'sage.structure.sage_object.SAGEObject'>
1664 ..... <type 'object'>
```

1665 4.12 ”は一つの”関係

1666 継承の概念の別の側面とは、親が出来ることなら何でも子供の類でも出来るので、親の類が
1667 使えるどの様な場所でも子供の類が使えます。整数類(Integer class)の継承階層に注目して
1668 みましょう。この階層は、整数は EuclideanDomainElement **であり**、EuclideanDomainElement
1669 は PrincipalIdealDomainElement **であり**、PrincipalIdealDomainElement は DedekindDomainElement
1670 **である**等となっています。最後に辿り着く SAGEObject **は一つの**対象です(丁度、SAGE での他
1671 の殆ど全ての類の様に object 類が根本の類で、全てはそこからの子孫になるからです。)
1672 一般的には、その先祖が使える任意の場所で子孫の類が使えるということです。

1673 4.13 こんがらがったかな?

1674 この章は多分貴方を困惑させるものかもしれませんが、そのことに悩むことはありません。この本
1675 の残りは SAGE で対象がどのように使われているかを示す例を含んでおり、たくさんの用いら
1676 れている対象を見て、それらのお陰でより快適になることでしょう。

1677

5 いろいろなこと

1678

5.1 前回の処理結果の参照

1679 一つのワークシートに複数のセルを広げて問題を解くとき、以前の処理結果を参照する事が
1680 望ましい事がしばしばあります。下線記号'_'は次の例で示すようにこの目的に使えます：

```
1681 2 + 3
1682 |
1683     5
1684 ---
1685 |
1686     5
```

```
1687 _ + 6
1688 |
1689     11
```

```
1690 a = _ * 2
1691 a
1692 |
1693     22
```

1694

5.2 例外処理

1695 SAGE のプログラムが動作している間に生じるかもしれない例外条件の処理を行う為に単一の
1696 方法を持っている事を保証する為、例外表示と処理の仕組は SAGE に組み込まれています。こ
1697 の節では単に表示された例外についてのみ解説します。何故なら、例外処理はこの文書の領
1698 域を越える進んだ話題だからです。

1699 次のプログラムは例外が生じる原因となり、それから、その例外の情報が表示されます：

```
1700 1/0
1701 |
1702     Exception (click to the left for traceback):
1703     ...
1704     ZeroDivisionError: Rational division by zero
```

1705 何故なら、1/0 は未定義の数学的演算なので、SAGE はその計算の実行が出来ません。そのプ
1706 ログラムの実行を停止し、この問題に関してプログラムの別の領域、或いは、利用者に報せ

1707 る例外を生成します。その例外を扱うプログラムの別部分がなければ、その例外のテキスト
1708 による説明が表示されます。この場合、その例外が利用者に ZeroDivisionError が生じたこ
1709 とと、これが”rational division by zero(零による有理数の割算)” を実行しようとした為
1710 に生じたことを利用者に報せています。

1711 とにかく、これは利用者にとってソースコードにある問題を突き止めて修正するのに十分な
1712 情報です。時には、利用者はその問題を突き止める為により多くの情報を必要とすることも
1713 あるので、例外はマウスを表示された例外のテキストの左側をクリックすれば、追加の情報
1714 が表示される様になっています。

```
1715     Traceback (most recent call last):
1716         File "", line 1, in
1717         File "/home/sage/sage_notebook/worksheets/tkosan/2/code/2.py",
1718             line 4, in
1719             Integer(1)/Integer(0)
1720         File "/opt/sage-2.8.3-linux-32bit-debian-4.0-i686-
1721             Linux/data/extcode/sage/", line 1, in
1722
1723             File "element.pyx", line 1471, in element.RingElement.__div__
1724             File "element.pyx", line 1485, in element.RingElement._div_c
1725             File "integer.pyx", line 735, in integer.Integer._div_c_impl
1726             File "integer_ring.pyx", line 185, in
1727             integer_ring.IntegerRing_class._div
1728             ZeroDivisionError: Rational division by zero
```

1729 この追加情報で、例外が生じた時に利用されている SAGE ライブラリの全てのプログラムの追
1730 跡結果とそのプログラムが含まれているファイル名が見られます。そのお陰で、例外が SAGE
1731 の虫か入力されたプログラムの虫を原因とするものかどうかを決定する為に、SAGE の猛者は
1732 ソースコードを見ることが出来ます。

1733 5.3 数値結果を得る

1734 対象の数値近似が必要になることがあります。SAGE はこれを遂行するいくつかの方法があ
1735 ります。一つの方法は `n()` 関数で、別のもう一つの方法は `n()` メソッドを使うことです。次の
1736 例では双方を利用した物を示します：

```
1737
1738 a = 3/4
1739 print a
1740 print n(a)
1741 print a.n()
1742 |
1743     3/4
```

```
1744 0.7500000000000000
1745 0.7500000000000000
```

1746 返却される対象の桁数は `digits` 助変数を使って調整で来ます :

```
1747 a = 3/4
1748 print a.n(digits=30)
1749 |
1750 0.75000000000000000000000000000000
```

1751 そして、精度の桁数は `prec` 助変数を使って調整で来ます :

```
1752 a = 4/3
1753 print a.n(prec=2)
1754 print a.n(prec=3)
1755 print a.n(prec=4)
1756 print a.n(prec=10)
1757 print a.n(prec=20)
1758 |
1759 1.5
1760 1.2
1761 1.4
1762 1.3
1763 1.3333
```

1764 5.4 式の表記指南

1765 常に、次の二項演算子には両方に空行一つを置きます:

1766 割当 '=' , 引数付き割当 ('+=', '-=', 等), 比較 ('==', '<', '>', '!=', '<>', '<=', '>=', 'in', 'not in',
1767 'is', 'is not'), 論理値 ('and', 'or', 'not').

1768 算術演算子 '+' と '-' のまわりには空行を置きますが、算術演算子 '*', '/', '%', と '^' には空行を置
1769 いてはいけません:

```
1770 x = x + 1
1771 x = x*3 - 5%2
1772 c = (a + b)/(a - b)
```

1773 添字引数や引数の附置として指す場合には等号 '=' の回りに空行を置かないように:

1774 `a.n(digits=5)`

1775 5.5 組込定数

1776 SAGE は組込の幾つかの数学定数を有しており、次は最も一般に使われるもののリストです:

1777 `Pi, pi`: 円周率

1778 `E, e`: 自然底

1779 `I, i`: 純虚数

1780

1781 `log2`: 実数 2 を底とする対数

1782 `Infinity, infinity`: 正や負の無限大を指す為に + や - を前に置けます。

1783 次の例では定数の利用が示されています:

1784 `a = pi.n()`

1785 `b = e.n()`

1786 `c = i.n()`

1787 `a, b, c`

1788 |

1789 `(3.14159265358979, 2.71828182845905, 1.00000000000000*I)`

1790 `r = 4`

1791 `a = 2*pi*r`

1792 `a, a.n()`

1793 |

1794 `(8*pi, 25.1327412287183)`

1795 SAGE での定数は大域変数として定義され、この**大域変数**は大半の SAGE のプログラムから、
1796 関数の内部やメソッドも含めて参照出来ます。定数は単純に、それらに一定の対象が割り当
1797 てられた変数なので、必要であっても定数対象が失われているのであれば、その変数に最割
1798 り当てが可能です。もし、変数にそれが通常持っている定数を再度割り当てなければなら
1799 ない場合は `restore()` 関数が用いられます。 次のプログラムではどの様に変数 `pi` がそれに

1800 割り当てられた対象 `7` を持つことができ、それから、`restore()` 関数にその名前を単引用符
1801 で括って引き渡すことで再びそのデフォルトの定数値を持つようになるかを示しています：

```
1802 print pi.n()
1803 pi = 7
1804 print pi
1805 restore('pi')
1806 print pi.n()
1807 |
1808     3.14159265358979
1809     7
1810     3.14159265358979
```

1811 `restore()` 関数が引数なしで呼び出されると、一度別の値を割り当てられた全ての定数は元の
1812 値に戻されます。

1813 5.6 根

1814 `sqrtrt()` 関数は値の平方根を得る為に使えるだけではなく、より一般的な手法が値の他の根を
1815 得る為に用いられています。例えば、8 の三乗根を得たければ：

$$\sqrt[3]{8}$$

1816 8 would be raised to the 1/3 power:

```
1817 8^(1/3)
1818 |
1819     2
```

1820 演算子の順序により、有理数 `1/3` は指数として評価されるために括弧で括る必要がありま
1821 す。

1822 5.7 記号変数

1823 ここまでで、我々が利用している全ての変数は割当のときに生成されています。たとえば、
1824 次の処理では変数 `w` が生成されて、数 `8` がそれに割り当てられます：

```
1825 w = 7
1826 w
```



```
1827 |
1828 7
```

1829 しかし、指定した値を何も割り当てていない変数进行处理する必要があるでしょう
1830 か？次の処理では変数 z の値を表示させようとしています。しかし、 z にはまだ値が割り当てら
1831 れていないので、例外処理が返されてしまいます：

```
1832 print z
1833 |
1834 Exception (click to the left for traceback):
1835 ...
1836 NameError: name 'z' is not defined
```

1837 数学で、“割り当てられていない変数”はいつでも使われます。SAGE は数学指向のソフトウエ
1838 アなので、割り当てられていない変数を使って処理する能力があります。SAGE では、割り当
1839 てられていない変数は**記号変数**と呼ばれ、それらは `var()` 函数を用いて定義されます。ワー
1840 クシートが最初に開かれた時に、変数 x は自律的に記号変数として定義され、貴方のコード
1841 の中で別の値を割り当てるまではそのままです。

1842 次のコードが新しく開いたワークシート上で実行されました：

```
1843 print x
1844 type(x)
1845 |
1846 x
1847 <class 'sage.calculus.calculus.SymbolicVariable'>
```

1848 変数 x には `SymbolicVariable` 型の対象が自律的に SAGE 環境によって割り当てられる事に
1849 注意してください。

1850 もし、記号変数として y と z も使いたければ、`var()` 函数をこの為に使わなければなりません。
1851 `var('x,y')`、または `var('x y')` の何れかが入力出来ます。`var()` 函数は一つ、または
1852 それ以上の変数名を文字列の中に受け入れられる様に設計されており、その名前はコンマか
1853 空行の何れかで分離することが出来ます。

1854 次のプログラムでは `var()` が y と z を記号変数として初期化することに用いている事を示し
1855 ます：

```
1856 var('y,z')
```

```
1857 y, z
1858 |
1859 (y, z)
```

1860 ひとつ、あるいはそれ以上の記号変数が定義された後で、reset()関数はそれらを未定義にす
1861 ることができます：

```
1862 reset('y,z')
1863 y, z
1864 |
1865 Exception (click to the left for traceback):
1866 ...
1867 NameError: name 'y' is not defined
```

1868 5.8 記号変数

1869 記号変数を含む式の事を記号式(symbolic expressions)と呼びます。次の例で、bは記
1870 号変数として定義されており、それから、記号式 $2*b$ の生成で用いられています。

```
1871 var('b')
1872 type(2*b)
1873 |
1874 <class 'sage.calculus.calculus.SymbolicArithmetic'>
```

1875 この例で見られる様に。記号式 $2*b$ はSymbolicArithmetics型の対象で置き換えられます。
1876 この式はまた変数に割り当てる事が可能です：

```
1877 m = 2*b
1878 type(m)
1879 |
1880 <class 'sage.calculus.calculus.SymbolicArithmetic'>
```

1881 次のプログラムは二つの記号式を生成し、それらを変数に割り当てて、それから、それらに
1882 関する演算を行います：

```
1883 m = 2*b
1884 n = 3*b
1885 m+n, m-n, m*n, m/n
1886 |
1887 (5*b, -b, 6*b^2, 2/3)
```

1888 二つの記号式同士の積のもう一つの別例です :

```
1889 m = 5 + b
1890 n = 8 + b
1891 y = m*n
1892 y
1893 |
1894 (b + 5)*(b + 8)
```

5.9 展開と因子分解

1895 前節の式の展開式が必要であれば、`expand()`メソッドを呼び出すことで簡単に得られます
1896 (この例では前節のセルが動作していると仮定しています) :

```
1897 z = y.expand()
1898 z
1899 |
1900 b^2 + 13*b + 40
```

1901 式の**展開**式は変数 `z` に割り当てられ、**因子分解**式は `z` から `factor()`メソッドから得られま
1902 す :

```
1903 z.factor()
1904 |
1905 (b + 5)*(b + 8)
```

1906 兎に角、数に変数に割当てをしなくても、回りを括弧で括って `factor()`メソッドを呼び出せ
1907 ば因子分解が行えます :

```
1908 (90).factor()
1909 |
1910 2 * 3^2 * 5
```

5.10 いろいろな記号式の例

```
1911 var('a,b,c')
```

```
1912 (5*a + b + 4*c) + (2*a + 3*b + c)
```

```
1913 |
```

```
1914 5*c + 4*b + 7*a
```

```
1915 (a + b) - (x + 2*b)
```

```
1916 |
```

```
1917 -x - b + a
```

```
1918 3*a^2 - a*(a -5)
```

```
1919 |
```

```
1920 3*a^2 - (a - 5)*a
```

```
1921 _.factor()
```

```
1922 |
```

```
1923 a*(2*a + 5)
```

5.11 記号式への値の引き渡し

1924 もし、値が記号式に引き渡されると、それは評価され、その結果が返却されます。もし、式
1925 が一つの変数を持っていれば、その値を次の様に単純に引き渡すことができます：

```
1926 a = x^2
```

```
1927 a(5)
```

```
1928 |
```

```
1929 25
```

1930 しかしながら、式が二つ以上の変数を持っていれば、各変数には名前を使って値を割り当て
1931 る必要があります：

```
1932 var('y')
```

```
1933 a = x^2 + y
```

```
1934 a(x=2, y=3)
```

```
1935 |
```

```
1936 7
```

1937 5.12 記号方程式と solve() 関数

1938 記号式の処理に加え、SAGE はまた記号方程式が扱えます：

```
1939 var('a')
1940 type(x^2 == 16*a^2)
1941 |
1942 <class 'sage.calculus.equations.SymbolicEquation'>
```

1943 この例で示すことが出来るように、記号方程式 $x^2 == 16*a^2$ は二重等号'=='を使う必要
1944 があるので、単等号'='を次のように使って変数に割り当てることが出来ます：

```
1945 m = x^2 == 16*a^2
1946 m, type(m)
1947 |
1948 (x^2 == 16*a^2, <class 'sage.calculus.equations.SymbolicEquation'>)
```

1949 たくさんの方程式が solve() 関数を用いて代数的に解けます：

```
1950 solve(m, a)
1951 |
1952 [a == -x/4, a == x/4]
```

1953 solve() 関数の第 1 引数は記号方程式で、第 2 の引数は解くべき記号変数です。

1954 solve() 関数もまた同時に方程式を解けます：

```
1955 var('i1,i2,i3,v0')
1956 a = (i1 - i3)*2 + (i1 - i2)*5 + 10 - 25 == 0
1957 b = (i2 - i3)*3 + i2*1 - 10 + (i2 - i1)*5 == 0
1958 c = i3*14 + (i3 - i2)*3 + (i3 - i1)*2 - (-3*v0) == 0
1959 d = v0 == (i2 - i3)*3
1960 solve([a,b,c,d], i1,i2,i3,v0)
1961 |
1962 [[i1 == 4, i2 == 3, i3 == -1, v0 == 12]]
```

1963 一つ以上の方程式を solve() に与える場合、リストの形式で与えなければならないことに注
1964 意してください。

1965 5.13 記号数学関数

1966 SAGE には数学の構文を使った関数を定義する能力があります。次の例では x を変数の関数と
1967 して関数 f を定義する様子を示しておきます：

```
1968 f(x) = x^2
```

```
1969 f, type(f)
1970 |
1971     (x |--> x^2,
1972     <class'sage.calculus.calculus.CallableSymbolicExpression'>)
```

1973 この方法で生成された対象は CallableSymbolicExpression 型で、これらは次の例題で示され
1974 る様に呼出可能な事を意味します：

```
1975 f(4), f(50), f(.2)
1976 |
1977     (16, 2500, 0.0400000000000000010)
```

1978 ここでは複製の代わりに上の CallableSymbolicExpression を使います：

```
1979 a = 0
1980 while a <= 9:
1981     f(a)
1982     a = a + 1
1983 |
1984     0
1985     1
1986     4
1987     9
1988     16
1989     25
1990     36
1991     49
1992     64
1993     81
```

1994 次の例では前の例題で行った同じ作業を完遂しますが、より進んだ言語機能を用いる点で異
1995 なります：

```
1996 a = srange(10)
1997 a
1998 |
1999     [0, 1, 2, 3, 4, 5, 6, 7, 8, 9]
```

```
2000 for num in a:
2001     f(num)
2002 |
2003     0
2004     1
2005     4
2006     9
```

2007 16
2008 25
2009 36
2010 49
2011 64
2012 81

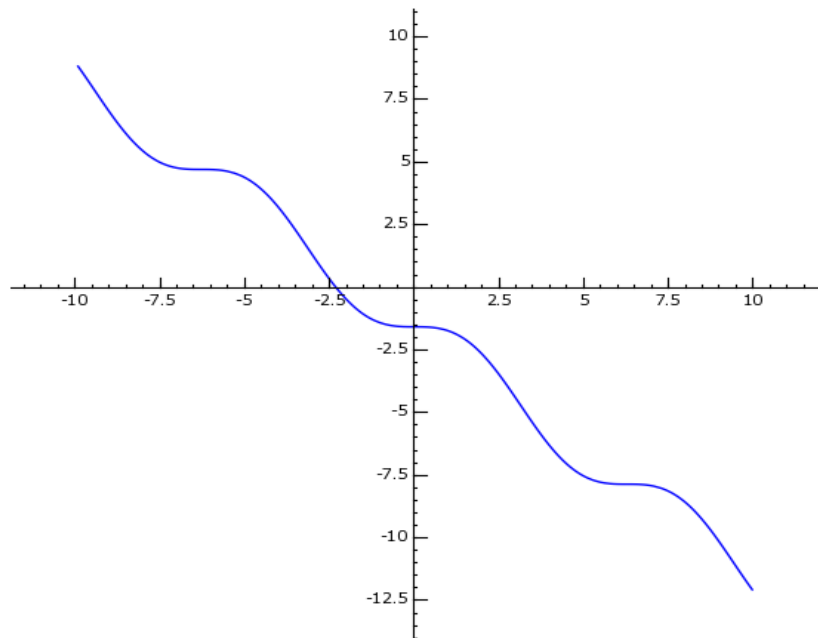
2013 5.14 グラフを使った根の検出と `find_root()` メソッドを利用した数 2014 値的な根の検出

2015 方程式が代数的に解けない場合には、`solve()` 関数は引き渡された入力の複製を返す事でこれ
2016 を示唆します。これは次の例で見られます：

```
2017 f(x) = sin(x) - x - pi/2  
2018 eqn = (f == 0)  
2019 solve(eqn, x)  
2020 |  
2021 [x == (2*sin(x) - pi)/2]
```

2022 ところで、代数的に解けない方程式はグラフを使ったり [数値的な](#) 方法の両方で解く事が出来
2023 ます。次の例では上の方程式をグラフを使って解いています：

```
2024 show(plot(f, -10, 10))  
2025 |
```



2026 このグラフは方程式の根が-2.5 よりも僅かに小さいことを示しています。

2027 次の例で、この方程式をより正確に `find_root()` メソッドを使って解けることを示してい
2028 ます：

```
2029 f.find_root(-10,10)
2030 |
2031     -2.309881460010057
```

2032 -10 と 10 を `find_root()` メソッドに引き渡す事で、根を探す区間を教えてください

2033 5.15 伝統的な書式で数学対象を表示

2034 早くから、SAGE は数学的対象をテキスト書式か伝統的書式の何れかで表示出来る様にして
2035 いました。この点について、我々は通常はテキスト書式を用います。数学対象を伝統的な書
2036 式で表示したければ、`show()` 関数が使えます。次の例題では数学式を生成し、それからテキ
2037 スト書式と伝統的書式の両方で表示します：

```
2038 var('y,b,c')
2039 z = (3*y^(2*b))/(4*x^c)^2
```

2040 #テキスト書式で式を表示します。

```
2041 z
2042 |
2043     3*y^(2*b)/(16*x^(2*c))
```

2044 #伝統的な書式で式を表示します。

```
2045 show(z)
2046 |
```

$$\frac{3 \cdot y^{2 \cdot b}}{16 \cdot x^{2 \cdot c}}$$

5.15.1 伝統的な数式で対象を表示する為に LaTeX を利用

2047 LaTeX (ラテフと発音、 <http://en.wikipedia.org/wiki/LaTeX>) は文書作成言語で、莫大な数
2048 学記号を扱う事が出来ます。SAGE の対象は `latex()` メソッドが呼び出されると、対応する
2049 LaTeX 表記を出力します。対象の LaTeX 表記は `latex()` 関数からも得られます：

```
2050 a = (2*x^2)/7
```



```
2051 latex(a)
2052 |
2053 \frac{{2 \cdot {x}^{2} }}{7}
```

2054 この結果を LaTeX に与えると、次に示す様な伝統的な数式出力を生成します：

$$\frac{2x^2}{7}$$

2055 図 2.5 で参照されている jsMath パッケージは SAGE のノートブックが LaTeX 入力を伝統的
2056 な数式に変換する為に用いているソフトウェアです。

2057 5.16 集合

2058 次の例題では SAGE が集合上で行える操作を示しています：

```
2059 a = Set([0,1,2,3,4])
2060 b = Set([5,6,7,8,9,0])
2061 a,b
2062 |
2063 ({0, 1, 2, 3, 4}, {0, 5, 6, 7, 8, 9})

2064 a.cardinality()
2065 |
2066 5

2067 3 in a
2068 |
2069 True

2070 3 in b
2071 |
2072 False

2073 a.union(b)
2074 |
2075 {0, 1, 2, 3, 4, 5, 6, 7, 8, 9}

2076 a.intersection(b)
2077 |
2078 {0}
```

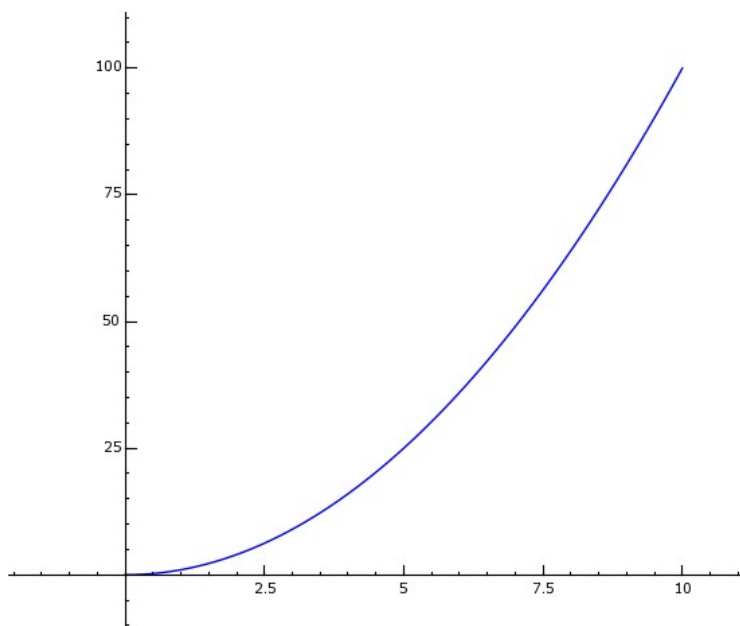
2079 **6 2D 描画**2080 **6.1 plot()と show()函数**

2081 SAGE は数学函数の 2D 描画を生成する為の沢山の方法を提供し、そのうちの 하나가 `plot()` 函
2082 数を `show()` 函数と絡めて用いる事です。次の例題では記号式を `plot()` 函数の最初の引数と
2083 して引き渡されることを示しています。第 2 の引数は X 軸上での描画の始点を指し、第 3 の
2084 引数が描画の終点を指します：

```
2085 a = x^2
2086 b = plot(a, 0, 10)
2087 type(b)
2088 |
2089 <class 'sage.plot.plot.Graphics'>
```

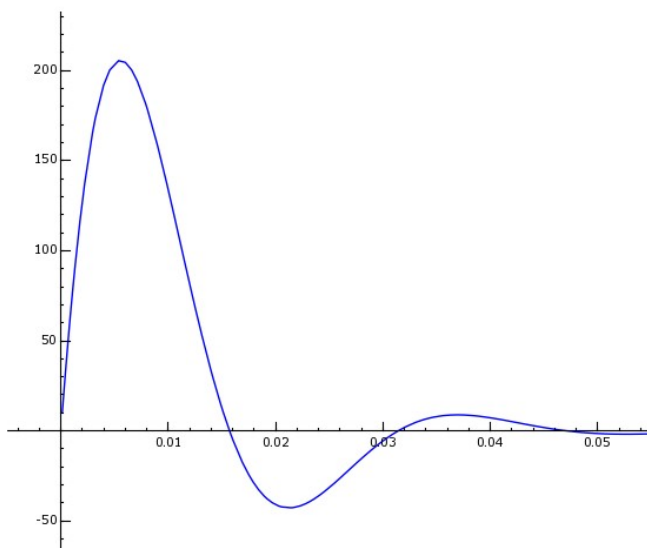
2090 `plot()` 函数は描画を表示する函数ではないことに注意してください。代わりに、それは
2091 `sage.plot.plot.Graphics` 型の対象を生成しますが、この対象は描画の与件を含んでいま
2092 す。`show()` 函数がこの描画の表示で使えるのです：

```
2093 show(b)
2094 |
```



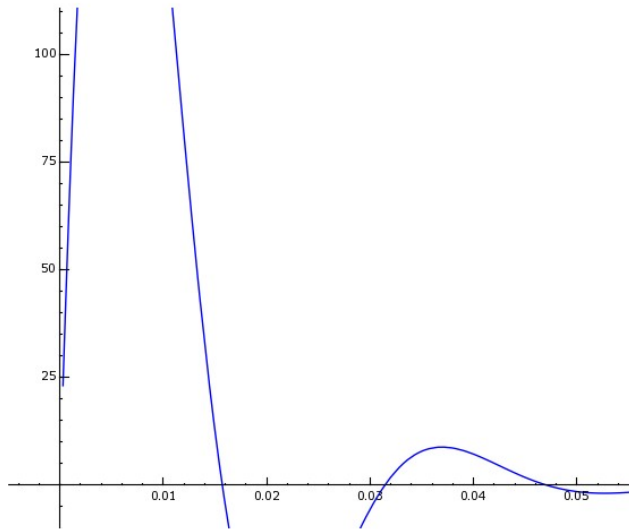
2095 `show()` 関数は `xmin`, `xmax`, `ymin` と `ymax` と呼ばれる 4 個の引数を持ち、これらは表示する
2096 描画を調整するために用いられます。 `figsize` 助変数もあり、これは画像の大きさを決定し
2097 ます。次の例では、 `xmin` と `xmax` を X 軸上の 0 と .05 の間の描画を表示する様に用いていま
2098 す。 `plot()` 関数は入力の労を減らす為に `show()` 関数への第 1 引数として用いる事が可能です
2099 (註: x 以外の任意の記号変数が用いられていれば、最初に `var()` 関数を使って宣言を行って
2100 いなければなりません):

```
2101 v = 400*e^(-100*x)*sin(200*x)
2102 show(plot(v,0,.1),xmin=0, xmax=.05, figsize=[3,3])
2103 |
```



2104 引数 `ymin` と `ymax` は上の描画で表示される y 軸の大きさを調整する為に使えます:

```
2105 show(plot(v,0,.1),xmin=0, xmax=.05, ymin=0, ymax=100, figsize=[3,3])
2106 |
```



6.1.1 描画の結合と描画の色の変更

2107 一つ、あるいはそれ以上の描画を纏めて一つの描画にする必要があるかもしれません。次の
 2108 例では、`show()` 関数を用いて 6 個の描画を結合します：

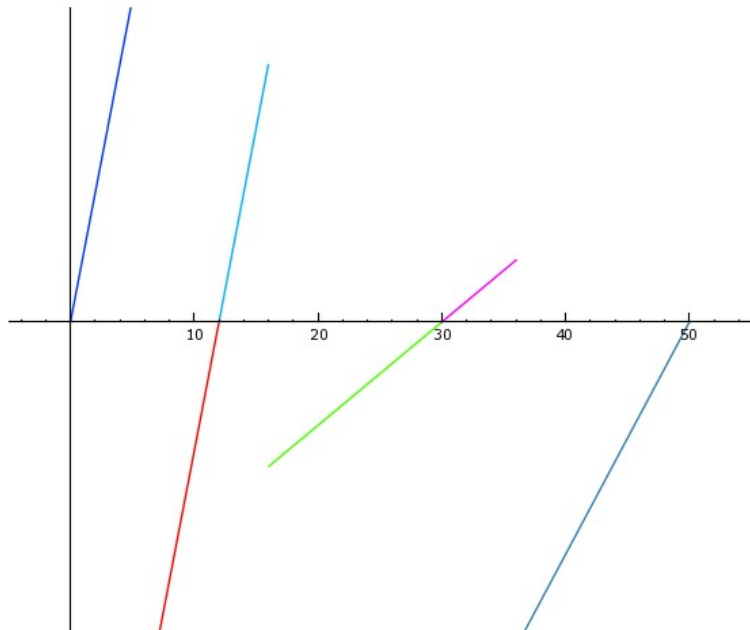
```

2109 var('t')
2110 p1 = t/4E5
2111 p2 = (5*(t - 8)/2 - 10)/1000000
2112 p3 = (t - 12)/400000
2113 p4 = 0.0000004*(t - 30)
2114 p5 = 0.0000004*(t - 30)
2115 p6 = -0.0000006*(6 - 3*(t - 46)/2)

2116 g1 = plot(p1, 0, 6, rgbcolor=(0, .2, 1))
2117 g2 = plot(p2, 6, 12, rgbcolor=(1, 0, 0))
2118 g3 = plot(p3, 12, 16, rgbcolor=(0, .7, 1))
2119 g4 = plot(p4, 16, 30, rgbcolor=(.3, 1, 0))
2120 g5 = plot(p5, 30, 36, rgbcolor=(1, 0, 1))
2121 g6 = plot(p6, 36, 50, rgbcolor=(.2, .5, .7))

2122 show(g1+g2+g3+g4+g5+g6, xmin=0, xmax=50, ymin=-.00001, ymax=.00001)
2123 |

```



2124 各描画の色は `rgbcolor` 助変数を用いて変更する事が出来ることに注意してください。RGB
 2125 は、赤、緑、青のタプルで成り立ち、`rgbcolor` 助変数には0から1の間の三つの値を割り
 2126 当てます。最初の値は最初の値は、描画の赤をどの程度(0から100%の間)似するかを指定
 2127 し、二番目の値は描画の緑をどの程度にするかを指定し、それから三番目の値は描画の青を
 2128 どの程度にするかを指定します。

6.1.2 グラフィックス対象とグラフィックスの結合

2129 一つの画像に様々な種類のグラフィックスを纏めることはしばしば便利な事です。次の例で
 2130 は、6点が描画されて、各点にはラベルがあります：

2131 `"""`

2132 次の点をグラフに描画：

2133 A (0,0)

2134 B (9,23)

2135 C (-15,20)

2136 D (22,-12)

2137 E (-5,-12)

2138 F (-22,-4)

2139 `"""`

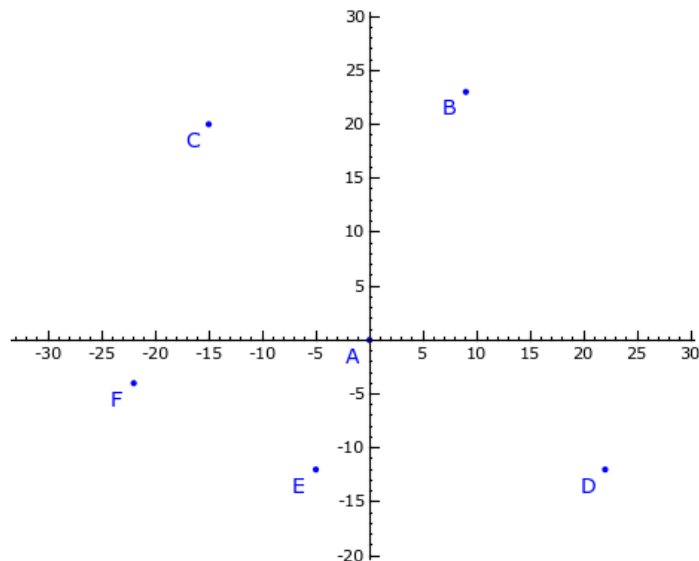
2140 # 複数のグラフィックス対象を持足せるためにグラフィックス対象を生成します。

```
2141 # これらのグラフィックス対象は同じ画像上に表示されます。
2142 g = Graphics()

2143 # 点のリストを生成し、それらをグラフィックス対象に追加します。
2144 points=[(0,0), (9,23), (-15,20), (22,-12), (-5,-12), (-22,-4)]
2145 g += point(points)

2146 # 点のラベルをグラフィックス対象に追加します。
2147 for (pnt,letter) in zip(points,['A','B','C','D','E','F']):
2148     g += text(letter,(pnt[0]-1.5, pnt[1]-1.5))

2149 #結合したグラフィックス対象を表示します。
2150 show(g,figsize=[5,4])
2151 |
```



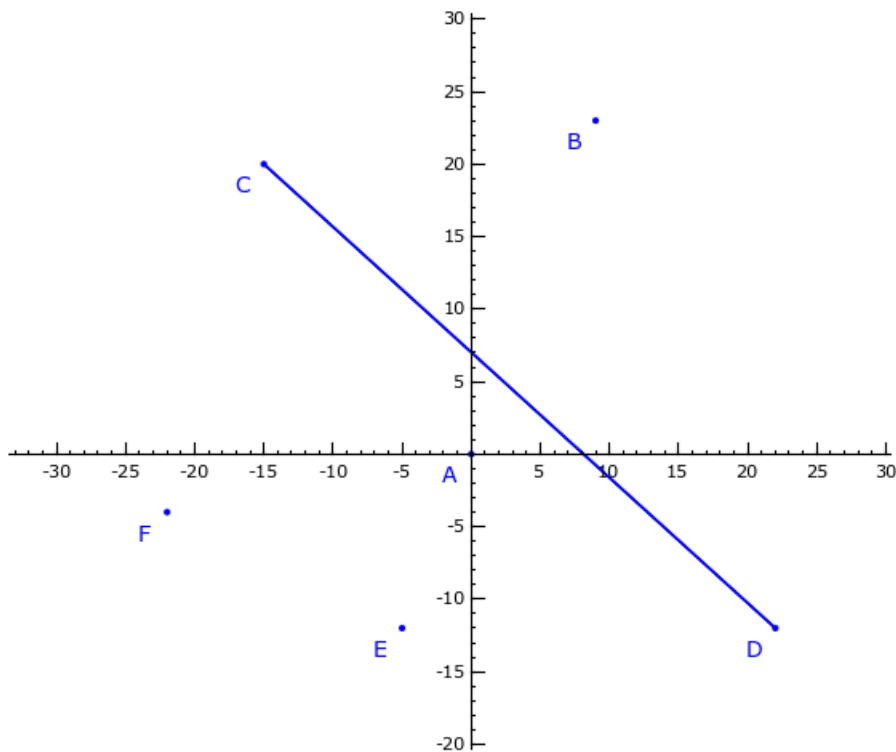
2152 最初に、空のグラフィックス対象を構築し、描画点のリストを `point()` 関数を使って生成し
2153 ます。それから、これらの描画される点は `+=` 演算子を使ってグラフィックス対象に追加され
2154 ます。次に、各点のラベルはグラフィックス対象に `for` 文を使って追加されます。最後に、
2155 グラフィックス対象は `show()` 関数を使ってワークシート上に表示されます。

2156 表示し終わっても、グラフィックス対象はそれの中に置かれた全てのグラフィックスを含ん
2157 であり、その上、必要であればグラフィックスをさらに追加することが可能です。たとえ
2158 ば、点 C と D の間の線分を描く必要があれば、これを完遂する為に別のセルで次のコードを
2159 実行することが出来ます：

```
2160 g += line([(-15,20), (22,-12)])
```

```
2161 show(g)
```

```
2162 |
```



2163 6.2 matplotlibによる進んだ描画

2164 SAGEはmatplotlib (<http://matplotlib.sourceforge.net>)ライブラリを必要であれば描画
2165 で用い、もし、plot()が提供する機能以上の描画制御が必要になれば、matplotlibの機能を直
2166 接使うことができます。matplotlibがどのように動作するかの詳細な解説はこの本の程度を
2167 越えてしまうので、この節では貴方の手助けになりそうな例を示しておきます。

6.2.1 網目と軸のラベルを持ったリストデータの描画

```
2168 x = [1921, 1923, 1925, 1927, 1929, 1931, 1933]
```

```
2169 y = [ .05, .6, 4.0, 7.0, 12.0, 15.5, 18.5]
```

```
2170 from matplotlib.backends.backend_agg import FigureCanvasAgg as \
```

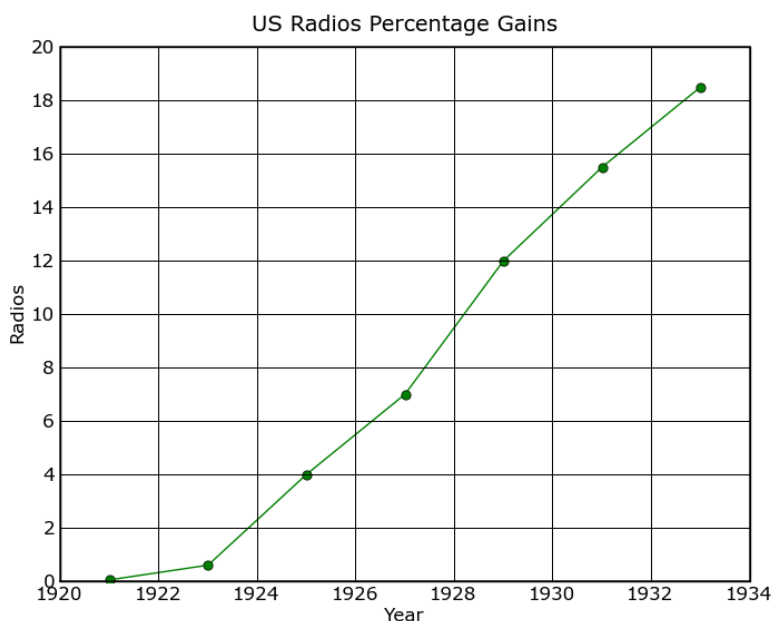
```
2171 FigureCanvas
```

```
2172 from matplotlib.figure import Figure
```

```
2173 from matplotlib.ticker import *
```

```
2174 fig = Figure()
```

```
2175 canvas = FigureCanvas(fig)
2176 ax = fig.add_subplot(111)
2177 ax.xaxis.set_major_formatter( FormatStrFormatter( '%d' ))
2178 ax.yaxis.set_major_locator( MaxNLocator(10) )
2179 ax.yaxis.set_major_formatter( FormatStrFormatter( '%d' ))
2180 ax.yaxis.grid(True, linestyle='-', which='minor')
2181 ax.grid(True, linestyle='-', linewidth=.5)
2182 ax.set_title('US Radios Percentage Gains')
2183 ax.set_xlabel('Year')
2184 ax.set_ylabel('Radios')
2185 ax.plot(x,y, 'go-', linewidth=1.0 )
2186 canvas.print_figure('ex1_linear.png')
2187 |
```



6.2.2 対数目盛の Y 軸を持った描画

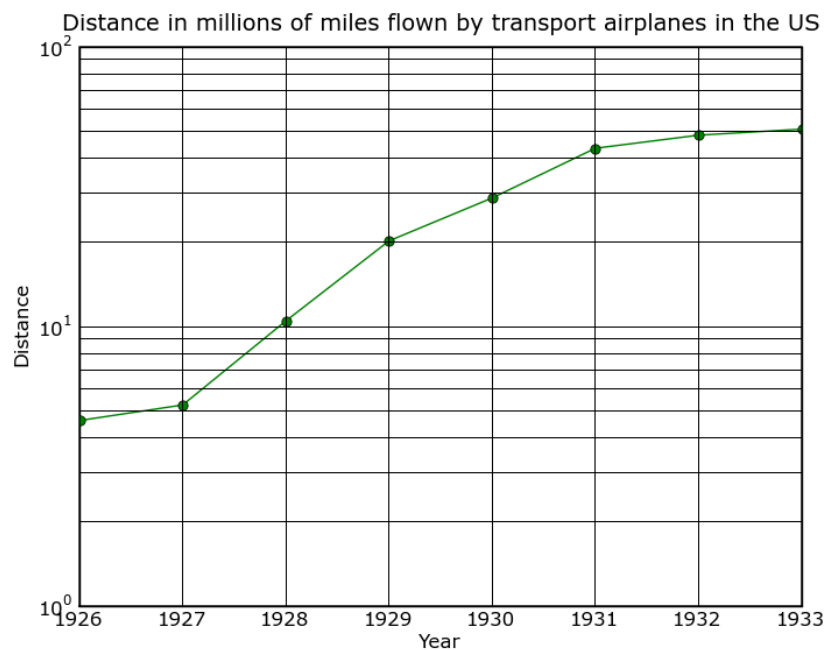
```
2188 x = [1926, 1927, 1928, 1929, 1930, 1931, 1932, 1933]
2189 y = [ 4.61, 5.24, 10.47, 20.24, 28.83, 43.40, 48.34, 50.80]
2190 from matplotlib.backends.backend_agg import FigureCanvasAgg as \
2191 FigureCanvas
2192 from matplotlib.figure import Figure
2193 from matplotlib.ticker import *
2194 fig = Figure()
2195 canvas = FigureCanvas(fig)
2196 ax = fig.add_subplot(111)
```



```

2197 ax.xaxis.set_major_formatter( FormatStrFormatter( '%d' ))
2198 ax.yaxis.set_major_locator( MaxNLocator(10) )
2199 ax.yaxis.set_major_formatter( FormatStrFormatter( '%d' ))
2200 ax.yaxis.grid(True, linestyle='-', which='minor')
2201 ax.grid(True, linestyle='-', linewidth=.5)
2202 ax.set_title('Distance in millions of miles flown by transport
2203 airplanes in the US')
2204 ax.set_xlabel('Year')
2205 ax.set_ylabel('Distance')
2206 ax.semilogy(x,y, 'go-', linewidth=1.0 )
2207 canvas.print_figure('ex2_log.png')
2208 |

```



6.2.3 描画の中にラベル付きの二つのグラフ

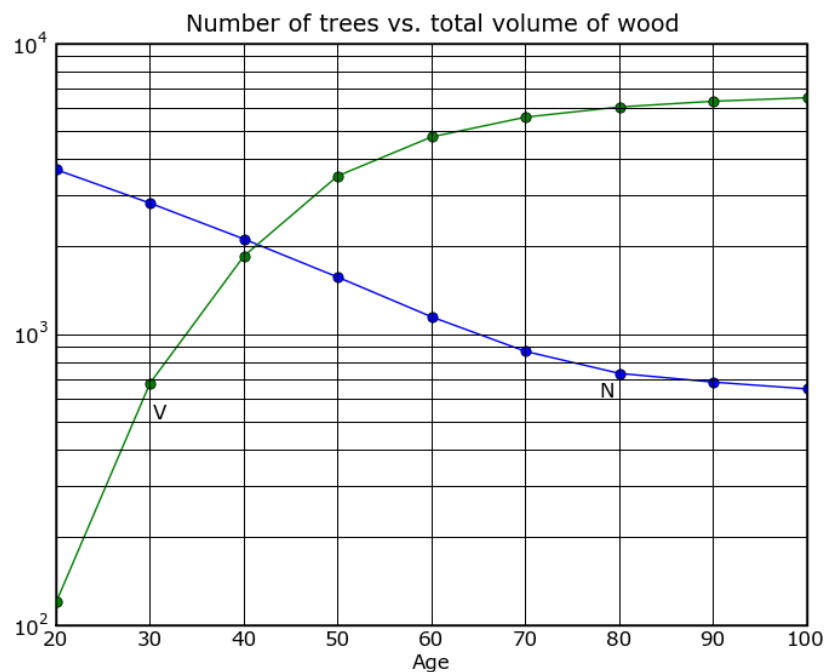
```

2209 x = [20, 30, 40, 50, 60, 70, 80, 90, 100]
2210 y = [3690, 2830, 2130, 1575, 1150, 875, 735, 686, 650]
2211 z = [120, 680, 1860, 3510, 4780, 5590, 6060, 6340, 6520]

2212 from matplotlib.backends.backend_agg import FigureCanvasAgg as \
2213 FigureCanvas
2214 from matplotlib.figure import Figure
2215 from matplotlib.ticker import *
2216 from matplotlib.dates import *
2217 fig = Figure()

```

```
2218 canvas = FigureCanvas(fig)
2219 ax = fig.add_subplot(111)
2220 ax.xaxis.set_major_formatter( FormatStrFormatter( '%d' ) )
2221 ax.yaxis.set_major_locator( MaxNLocator(10) )
2222 ax.yaxis.set_major_formatter( FormatStrFormatter( '%d' ) )
2223 ax.yaxis.grid(True, linestyle='-', which='minor')
2224 ax.grid(True, linestyle='-', linewidth=.5)
2225 ax.set_title('Number of trees vs. total volume of wood')
2226 ax.set_xlabel('Age')
2227 ax.set_ylabel('')
2228 ax.semilogy(x,y, 'bo-', linewidth=1.0 )
2229 ax.semilogy(x,z, 'go-', linewidth=1.0 )
2230 ax.annotate('N', xy=(550, 248), xycoords='figure pixels')
2231 ax.annotate('V', xy=(180, 230), xycoords='figure pixels')
2232 canvas.print_figure('ex5_log.png')
2233 |
```



2234 7 SAGE の書式

2235 SAGE はとても柔軟な環境なので、それ故に様々な利用方法があります。この章では二つの
2236 SAGE の構文書式について議論しますが、それらは**スピード書式(Speed Usage Style)**と
2237 **OpenOffice プレゼン書式(OpenOffice Presentation Style)**と呼びます。

2238 スピード書式は問題を可能な限り素早く解くことを目的に設計されており、結果の見栄えを
2239 良くする為に割く労力を最低にしています。この書式は通常の数学の教科書にある章末問題
2240 を解く事にとりわけ適しています。

2241 OpenOffice プレゼン書式は数学文書の作成能力がない人でも最小の労力で数学文書か書ける
2242 様に設計されています。このプレゼン書式はお家での宿題、報告書、記事や本等の作成に便
2243 利で、この本はこの書式を使って構築したものです。

2244 7.1 スピード書式

2245 (まだ...)

2246 7.2 オープンオフィスプレゼン書式

2247 (まだ...)

2248 **8 高校数学の問題（大半がまだ著作中）**2249 **8.1 Pre-Algebra**

Wikipedia entry.	http://en.wikipedia.org/wiki/Pre-algebra
------------------	---

2250 (In development...)

8.1.1 方程式

Wikipedia entry.	http://en.wikipedia.org/wiki/Equation
------------------	---

2251 (In development...)

8.1.2 式

Wikipedia entry.	http://en.wikipedia.org/wiki/Mathematical_expression
------------------	---

2252 (In development...)

8.1.3 幾何学

Wikipedia entry.	http://en.wikipedia.org/wiki/Geometry
------------------	---

2253 (In development...)

8.1.4 不等式

Wikipedia entry.	http://en.wikipedia.org/wiki/Inequality
------------------	---

2254 (In development...)

8.1.5 線形函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Linear_functions
------------------	---

2255 (In development...)

8.1.6 Measurement

Wikipedia entry.	http://en.wikipedia.org/wiki/Measurement
------------------	---

2256 (In development...)

8.1.7 非線形方程式

Wikipedia entry.	http://en.wikipedia.org/wiki/Nonlinear_system
------------------	---

2257 (In development...)

8.1.8 Number Sense And Operations

Wikipedia entry.	http://en.wikipedia.org/wiki/Number_sense
------------------	---

Wikipedia entry.	http://en.wikipedia.org/wiki/Operation_(mathematics)
------------------	---

2258 (In development...)

2259 8.1.8.1 分数の約分

2260 """

2261 問題:

2262 90/105 を約分しなさい。

2263 解:

2264 この問題を解く一つの方法は、分子と分母の双方を素数因子に素因数分解して共通の因子を
2265 見つけて、それから分子と分母の双方をこれらの因子で割ってしまいます。

2266 """

2267 n = 90

2268 d = 105

2269 print n,n.factor()

2270 print d,d.factor()

2271 |

2272 Numerator: 2 * 3^2 * 5

2273 Denominator: 3 * 5 * 7

2274 """

2275 分母と分子の双方に因子 3 と 5 が現れる事が分かり、そこで、分子と分母の双方を 3*5 で割
2276 ります:

2277 """

2278 n2 = n/(3*5)

```

2279 d2 = d/(3*5)
2280 print "Numerator2:",n2
2281 print "Denominator2:",d2
2282 |
2283     Numerator2: 6
2284     Denominator2: 7

```

```

2285
2286 """
2287 それ故に、6/7 が 90/105 の約分になります。

```

2288 この問題はまた、単に 90/105 をセルの中に入れてだけでも解けます。何故なら、有理数対象
 2289 は自動的に約分されるからです：

```

2290 """
2291 90/105
2292 |
2293     6/7

```

8.1.9 多項式函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Polynomial_function
------------------	---

2294 (まだ...)

2295 8.2 代数

Wikipedia entry.	http://en.wikipedia.org/wiki/Algebra_1
------------------	---

2296 (まだ...)

8.2.1 絶対値函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Absolute_value
------------------	---

2297 (まだ...)

8.2.2 複素数

Wikipedia entry.	http://en.wikipedia.org/wiki/Complex_numbers
------------------	---

2298 (まだ...)

8.2.3 合成函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Composite function
------------------	---

2299 (まだ...)

8.2.4 Conics

Wikipedia entry.	http://en.wikipedia.org/wiki/Conics
------------------	---

2300 (まだ...)

8.2.5 データ解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Data analysis
------------------	---

2301 (まだ...)

9 離散数学：初等的数とグラフ理論

Wikipedia entry.	http://en.wikipedia.org/wiki/Discrete mathematics
------------------	---

2302 (まだ...)

9.1.1 方程式

Wikipedia entry.	http://en.wikipedia.org/wiki/Equation
------------------	---

2303 (まだ...)

2304 9.1.1.1 記号分数の約分

2305 """

2306 問題:

2307 式 $(6x^2 - b) / (b - 6ab)$ を約分しなさい。ここで a と b は正整数を表現します。

2308 解:

2309 """

2310 `var('a,b')`

2311 `n = 6*a^2 - a`

2312 `d = b - 6 * a * b`

```

2313 print n
2314 print "          _____"
2315 print d
2316 |
2317          2
2318        6 a  - a
2319        -----
2320        b - 6 a b

2321 """
2322 分子と分母の双方の因子分解を始め、それから共通因子を探します：
2323 """
2324 n2 = n.factor()
2325 d2 = d.factor()
2326 print "Factored numerator:",n2.__repr__()
2327 print "Factored denominator:",d2.__repr__()
2328 |
2329     Factored numerator: a*(6*a - 1)
2330     Factored denominator: -(6*a - 1)*b

2331 """
2332 最初に、分子と分母の双方に何も共通因子が現れません。ところで、分子をもっと調べる
2333 と、-1を掛けることで(1 - 6 a)が見つけれられるので、この(6 a - 1)が答で、この因子も分
2334 子に含まれています。だから、次の段階では分子と分母の双方に-1を掛けます：
2335 """
2336 n3 = n2 * -1
2337 d3 = d2 * -1
2338 print "Numerator * -1:",n3.__repr__()
2339 print "Denominator * -1:",d3.__repr__()
2340 |
2341     Numerator * -1: -a*(6*a - 1)
2342     Denominator * -1: (6*a - 1)*b

2343 """
2344 そこで、約分の為に分子と分母の双方が(6*a - 1)で割り切ることが出来ます：

```



```

2345 """
2346 common_factor = 6*a - 1
2347 n4 = n3 / common_factor
2348 d4 = d3 / common_factor
2349 print n4
2350 print "          ----"
2351 print d4
2352 |
2353          - a
2354          ---
2355          b

2356 """
2357 この問題は SymbolicArithmetic 対象を使うことで、より直接的に解けます：
2358 """
2359 z = n/d
2360 z.simplify_rational()
2361 |
2362     -a/b

```

2363 9.1.1.2 二つの記号分数の積を計算

2364 次の計算を実行： $\left(\frac{x}{2y}\right)^2 \cdot \left(\frac{4y^2}{3x}\right)^3$

```

2365 """
2366 記号式は通常自律的に簡易化されるので、この問題で実行されるべき全ての事は式を入力し
2367 てある変数に割り当てる事です：
2368 """

```

```

2369 var('y')
2370 a = (x/(2*y))^2 * ((4*y^2)/(3*x))^3

2371 #テキスト形式で式を表示：
2372 a
2373 |

```

2374 $16*y^4/(27*x)$

2375 #伝統的な書式で式を表示：

2376 show(a)

2377 |

$$\frac{16 \cdot y^4}{27 \cdot x}$$

2378 **9.1.1.3 x について線型方程式を解く**

2379 $3x+2x-8=5x-3x+7$ を解け

2380 """

2381 この方程式を SymbolicEquation 対象として置くと、項の様に自律的に結合されます：

2382 """

2383 $a = 5*x + 2*x - 8 == 5*x - 3*x + 7$

2384 a

2385 |

2386 $7*x - 8 == 2*x + 7$

2387 """

2388 最初に、方程式の左側に x の項を移動させる為に 2x を両側から引きます（註：下線記

2389 号'_' は直前に実行されたセルの結果を保持しています：

2390 """

2391 $_ - 2*x$

2392 |

2393 $5*x - 8 == 7$

2394 """

2395 両側に 8 を加えます：

2396 """

2397 $_ +8$

2398 |

2399 $5*x == 15$

2400 """

2401 最後に、解を決める為に両側を 5 で割ります :

2402 """

2403 $_ / 5$

2404 |

2405 $x == 3$

2406 """

2407 この問題は solve() を使うと自動的に解かれます :

2408 """

2409 solve(a, x)

2410 |

2411 $[x == 3]$

2412 **9.1.1.4 分数を持つ線型方程式の解法**

2413 $\frac{16x-13}{6} = \frac{3x+5}{2} - \frac{4-x}{3}$ を解け

2414 """

2415 最初の段階は方程式を SymbolicEquation 対象に置き換える事です。方程式を表示させる事

2416 は、方程式が正しく入力されたかを検証する事が出来るので良い思いつきです :

2417 """

2418 $a = (16*x - 13)/6 == (3*x + 5)/2 - (4 - x)/3$

2419 a

2420 |

2421 $(16*x - 13)/6 == (3*x + 5)/2 - (4 - x)/3$

2422 """

2423 テキストとして方程式がされると、方程式が正しく入力されたかどうか確認する事は困難で

2424 す。だから、伝統的な書式でも表示させましょう :

2425 """

2426 show(a)

2427 |

$$\frac{16 \cdot x - 13}{6} = \frac{3 \cdot x + 5}{2} - \frac{4 - x}{3}$$

2428 """

2429 次の段階で、方程式の分母を消せるように最小公倍数(LCD)を計算しましょう :

2430 """

```
2431 lcm([6, 2, 3])
2432 |
2433     6

2434 """
2435 方程式の LCD は 6 なので、6 を掛けて分数を消してしまいます：
2436 """
2437 b = a*6
2438 b
2439 |
2440     16*x - 13 == 6*((3*x + 5)/2 - (4 - x)/3)

2441 """
2442 方程式の右辺にはまだ分数があるので展開してみましよう：
2443 """
2444 c = b.expand()
2445 c
2446 |
2447     16*x - 13 == 11*x + 7

2448 """
2449 11x を方程式の左辺に移す為に、SymbolEquation から 11x を引きます：
2450 """
2451 d = c - 11*x
2452 d
2453 |
2454     5*x - 13 == 7

2455 """
2456 -13 を方程式の右辺に移動させるために SymbolEquation に対して 13 を加えましよう：
2457 """
2458 e = d + 13
2459 e
2460 |
2461     5*x == 20

2462 """
```

2463 最後に、SymbolEquation を 5 で割って、方程式の左側を x だけにすると解が得られます：

2464 """

2465 f = e / 5

2466 f

2467 |

2468 x == 4

2469 """

2470 この問題は solve() を使うと自律的に解かれます：

2471 """

2472 solve(a, x)

2473 |

2474 [x == 4]

9.1.2 指数関数

Wikipedia の見出

http://en.wikipedia.org/wiki/Exponential_function

2475 (まだ...)

9.1.3 冪

Wikipedia の見出.

<http://en.wikipedia.org/wiki/Exponent>

2476 (まだ...)

9.1.4 式

Wikipedia の見出.

[http://en.wikipedia.org/wiki/Expression_\(mathematics\)](http://en.wikipedia.org/wiki/Expression_(mathematics))

2477 (まだ...)

9.1.5 不等式

Wikipedia の見出.

<http://en.wikipedia.org/wiki/Inequality>

2478 (まだ...)

9.1.6 逆関数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Inverse_function
----------------	---

2479 (まだ...)

9.1.7 線型方程式と函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Linear_functions
----------------	---

2480 (まだ...)

9.1.8 線型プログラミング

Wikipedia の見出.	http://en.wikipedia.org/wiki/Linear_programming
----------------	---

2481 (まだ...)

9.1.9 対数函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Logarithmic_function
----------------	---

2482 (まだ...)

9.1.10 兵站函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Logistic_function
----------------	---

2483 (まだ...)

9.1.11 行列

Wikipedia の見出.	http://en.wikipedia.org/wiki/Matrix_(mathematics)
----------------	---

2484 (まだ...)

9.1.12 Parametric Equations

Wikipedia 見出.	http://en.wikipedia.org/wiki/Parametric_equation
---------------	---

2485 (In development...)

9.1.13 区分函数

Wikipedia 見出.	http://en.wikipedia.org/wiki/Piecewise_function
---------------	---

2486 (In development...)

9.1.14 多項式函数

Wikipedia 見出.	http://en.wikipedia.org/wiki/Polynomial_function
---------------	---

2487 (In development...)

9.1.15 冪級数函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Power_function
----------------	---

2488 (In development...)

9.1.16 Quadratic Functions

Wikipedia entry.	http://en.wikipedia.org/wiki/Quadratic_function
------------------	---

2489 (In development...)

9.1.17 Radical Functions

Wikipedia の見出.	http://en.wikipedia.org/wiki/Nth_root
----------------	---

2490 (まだ...)

9.1.18 有理函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Rational_function
----------------	---

2491 (In development...)

9.1.19 列

Wikipedia の見出.	http://en.wikipedia.org/wiki/Sequence
----------------	---

2492 (In development...)

9.1.20 級数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Series_mathematics
----------------	---

2493 (In development...)

9.1.21 方程式系

Wikipedia の見出.	http://en.wikipedia.org/wiki/System_of_equations
----------------	---

2494 (In development...)

9.1.22 変換

Wikipedia の見出.	http://en.wikipedia.org/wiki/Transformation_(geometry)
----------------	---

2495 (In development...)

9.1.23 三角函数

Wikipedia の見出.	http://en.wikipedia.org/wiki/Trigonometric_function
----------------	---

2496 (In development...)

2497 9.2 Precalculus And Trigonometry

Wikipedia entry.	http://en.wikipedia.org/wiki/Precalculus
	http://en.wikipedia.org/wiki/Trigonometry

2498 (In development...)

9.2.1 二項定理

Wikipedia entry.	http://en.wikipedia.org/wiki/Binomial_theorem
------------------	---

2499 (In development...)

9.2.2 複素数

Wikipedia entry.	http://en.wikipedia.org/wiki/Complex_numbers
------------------	---

2500 (In development...)

9.2.3 合成函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Composite_function
------------------	---

2501 (In development...)

9.2.4 Conics

Wikipedia entry.	http://en.wikipedia.org/wiki/Conics
------------------	---

2502 (In development...)

9.2.5 データ解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Data_analysis
------------------	---

2503 (In development...)

10 離散数学: Elementary Number とグラフ理論

Wikipedia entry.	http://en.wikipedia.org/wiki/Discrete_mathematics
------------------	---

2504 (In development...)

10.1.1 方程式

Wikipedia entry.	http://en.wikipedia.org/wiki/Equation
------------------	---

2505 (In development...)

10.1.2 指数函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Equation
------------------	---

2506 (In development...)

10.1.3 逆函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Inverse_function
------------------	---

2507 (In development...)

10.1.4 対数函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Logarithmic_function
------------------	---

2508 (In development...)

10.1.5 兵站函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Logistic_function
------------------	---

2509 (In development...)

10.1.6 行列と行列代数

Wikipedia entry.	http://en.wikipedia.org/wiki/Matrix_(mathematics)
------------------	---

2510 (In development...)

10.1.7 数学的解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Mathematical_analysis
------------------	---

2511 (In development...)

10.1.8 Parametric Equations

Wikipedia entry.	http://en.wikipedia.org/wiki/Parametric_equation
------------------	---

2512 (In development...)

10.1.9 区分函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Piecewise_function
------------------	---

2513 (In development...)

10.1.10 Polar Equations

Wikipedia entry.	http://en.wikipedia.org/wiki/Polar_equation
------------------	---

2514 (In development...)

10.1.11 多項式函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Polynomial_function
------------------	---

2515 (In development...)

10.1.12 冪関数

Wikipedia entry.	http://en.wikipedia.org/wiki/Power_function
------------------	---

2516 (In development...)

10.1.13 多項式函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Quadratic_function
------------------	---

2517 (In development...)

10.1.14 Radical Functions

Wikipedia entry.	http://en.wikipedia.org/wiki/Nth_root
------------------	---

2518 (In development...)

10.1.15 有理函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Rational_function
------------------	---

2519 (In development...)

10.1.16 実数

Wikipedia entry.	http://en.wikipedia.org/wiki/Real_number
------------------	---

2520 (In development...)

10.1.17 列

Wikipedia entry.	http://en.wikipedia.org/wiki/Sequence
------------------	---

2521 (In development...)

10.1.18 級数

Wikipedia entry.	http://en.wikipedia.org/wiki/Series_(mathematics)
------------------	---

2522 (In development...)

10.1.19 集合

Wikipedia entry.	http://en.wikipedia.org/wiki/Set
------------------	---

2523 (In development...)

10.1.20 方程式系

Wikipedia entry.	http://en.wikipedia.org/wiki/System_of_equations
------------------	---

2524 (In development...)

10.1.21 変換

Wikipedia entry.	http://en.wikipedia.org/wiki/Transformation_(geometry)
------------------	---

2525 (In development...)

10.1.22 三角函数

Wikipedia entry.	http://en.wikipedia.org/wiki/Trigonometric_function
------------------	---

2526 (In development...)

10.1.23 ベクトル

Wikipedia entry.	http://en.wikipedia.org/wiki/Vector
------------------	---

2527 (In development...)

2528 10.2 解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Calculus
------------------	---

2529 (In development...)

10.2.1 微分

Wikipedia entry.	http://en.wikipedia.org/wiki/Derivative
------------------	---

2530 (In development...)

10.2.2 積分

Wikipedia entry.	http://en.wikipedia.org/wiki/Integral
------------------	---

2531 (In development...)

10.2.3 極限

Wikipedia entry.	http://en.wikipedia.org/wiki/Limit_(mathematics)
------------------	---

2532 (In development...)

10.2.4 多項式近似と級数

Wikipedia entry.	http://en.wikipedia.org/wiki/Convergent_series
------------------	---

2533 (In development...)

2534 10.3 統計

Wikipedia entry.	http://en.wikipedia.org/wiki/Statistics
------------------	---

2535 (In development...)

10.3.1 データ解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Data_analysis
------------------	---

2536 (In development...)

10.3.2 Inferential Statistics

Wikipedia entry.	http://en.wikipedia.org/wiki/Inferential_statistics
------------------	---

2537 (In development...)

10.3.3 標準分布

Wikipedia entry.	http://en.wikipedia.org/wiki/Normal_distribution
------------------	---

2538 (In development...)

10.3.4 1変数解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Univariate
------------------	---

2539 (In development...)

10.3.5 確率と試行

Wikipedia entry.	http://en.wikipedia.org/wiki/Probability
------------------	---

2540 (In development...)

10.3.6 2変数解析

Wikipedia entry.	http://en.wikipedia.org/wiki/Multivariate
------------------	---

2541 (In development...)

2542 **11 高校生の科学の問題**

2543 (In development...)

2544 **11.1 物理学**

Wikipedia entry.	http://en.wikipedia.org/wiki/Physics
------------------	---

2545 (In development...)

11.1.1 原子物理

Wikipedia entry.	http://en.wikipedia.org/wiki/Atomic_physics
------------------	---

2546 (In development...)

11.1.2 円運動

Wikipedia entry.	http://en.wikipedia.org/wiki/Circular_motion
------------------	---

2547 (In development...)

11.1.3 力学

Wikipedia entry.	http://en.wikipedia.org/wiki/Dynamics_(physics)
------------------	---

2548 (In development...)

11.1.4 電磁界

Wikipedia entry.	http://en.wikipedia.org/wiki/Electricity
------------------	---

	http://en.wikipedia.org/wiki/Magnetism
--	---

2549 (In development...)

11.1.5 流体

Wikipedia entry.	http://en.wikipedia.org/wiki/Fluids
------------------	---

2550 (In development...)

11.1.6 運動学

2551	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Kinematics
------	---	---

11.1.7 光

2552	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Light
------	---	---

11.1.8 光学

2553	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Optics
------	---	---

11.1.9 相対性理論

2554	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Relativity
------	---	---

11.1.10 回転運動

2555	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Rotational_motion
------	---	---

11.1.11 音響

2556	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Sound
------	---	---

11.1.12 波

2557	Wikipedia entry. (In development...)	http://en.wikipedia.org/wiki/Waves
------	---	---

11.1.13 熱力学

Wikipedia entry.	http://en.wikipedia.org/wiki/Thermodynamics
------------------	---

2558 (In development...)

11.1.14 仕事

Wikipedia entry.	http://en.wikipedia.org/wiki/Mechanical_work
------------------	---

2559 (In development...)

11.1.15 エネルギー

Wikipedia entry.	http://en.wikipedia.org/wiki/Energy
------------------	---

2560 (In development...)

11.1.16 モーメント

Wikipedia entry.	http://en.wikipedia.org/wiki/Momentum
------------------	---

2561 (In development...)

11.1.17 Boiling

Wikipedia entry.	http://en.wikipedia.org/wiki/Boiling
------------------	---

2562 (In development...)

11.1.18 浮力

Wikipedia entry.	http://en.wikipedia.org/wiki/Bouyancy
------------------	---

2563 (In development...)

11.1.19 Convection

Wikipedia entry.	http://en.wikipedia.org/wiki/Convection
------------------	---

2564 (In development...)

11.1.20 密度

Wikipedia entry.	http://en.wikipedia.org/wiki/Density
------------------	---

2565 (In development...)

11.1.21 Diffusion

Wikipedia entry.	http://en.wikipedia.org/wiki/Diffusion
------------------	---

2566 (In development...)

11.1.22 Freezing

Wikipedia entry.	http://en.wikipedia.org/wiki/Freezing
------------------	---

2567 (In development...)

11.1.23 摩擦

Wikipedia entry.	http://en.wikipedia.org/wiki/Friction
------------------	---

2568 (In development...)

11.1.24 熱伝導

Wikipedia entry.	http://en.wikipedia.org/wiki/Heat_transfer
------------------	---

2569 (In development...)

11.1.25 Insulation

Wikipedia entry.	http://en.wikipedia.org/wiki/Insulation
------------------	---

2570 (In development...)

11.1.26 Newton の法則

Wikipedia entry.	http://en.wikipedia.org/wiki/Newtons_laws
------------------	---

2571 (In development...)

11.1.27 圧力

Wikipedia entry.	http://en.wikipedia.org/wiki/Pressure
------------------	---

2572 (In development...)

11.1.28 プーリー

Wikipedia entry.	http://en.wikipedia.org/wiki/Pulley
------------------	---

2573 (In development...)

2574 12 計算の基礎

2575 12.1 計算機ってなあに

2576 多くの人々は計算機が複雑である為に理解し難いと思っています。計算機は実際に複雑です
2577 が、これが理解し難くしている理由ではありません。計算機が理解し難いのは、実体のある
2578 世界には計算機の小さな部分しか存在しないからです。実体としての計算機は人間が見ること
2579 の出来る部分でしかなく、計算機の残りは不可視の実体の無い世界に存在するからです。
2580 この不可視の世界は概念（アイデア）の世界であり、計算機の大半は概念としてこの実体の無い
2581 世界にあるのです。

2582 計算機を理解する為の鍵は、概念に基づく機械の目的が全ての型の概念の自動操作である事
2583 を理解する事にあります。'計算機' という名前は計算機が現実は何であるかを描くには余り
2584 助けになるものではなく、寧ろ、概念操作機器(Idea Manipulation Device)、即ち、IMDと
2585 言う名前の方が良いでしょう。

2586 さて、概念は実体の無い対象なので、それらは実世界に持ち込めませんし、実世界の対象を
2587 概念の世界に持ち込む事も勿論出来ません。これらの二つの世界は互いに分離しているの
2588 で、実世界の対象は記号による遠隔制御で概念の世界の対象を操作する事になります。

2589 12.2 記号(symbol)って何?

2590 記号(symbol) は別の対象を表現する為に用いられる対象です。 図 12.1 は電話の記号の
2591 例で、この記号は実際の電話を表現する為に用いています。

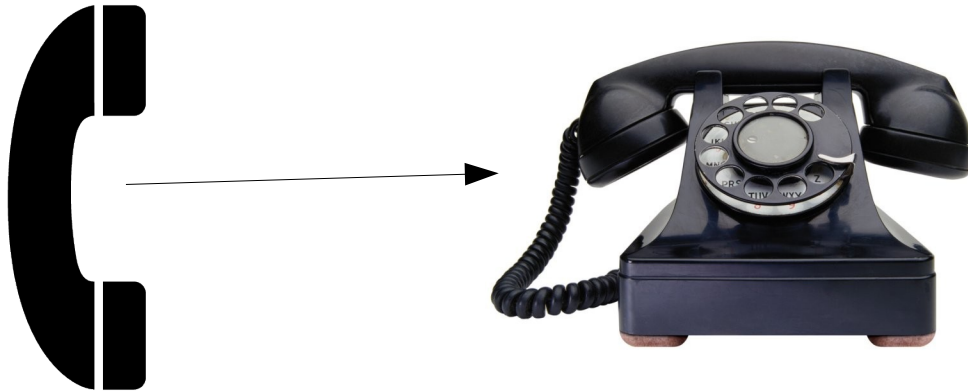


図 12.1: 実体を持つ対象と関連付けられた記号

2592 図 12.1 で示した電話の記号は通常、（紙の様な）平面上でインクで印刷される事で生成さ
2593 れています。一般的に、ある種の並びとして調整された実体を持つ物の任意の型（或いは、
2594 実体を持つ物の属性）は記号として用いる事が出来ます。

2595 12.3 記号としてビットの並びを使う計算機

2596 実体のある物から出来上がった記号は全ての実体のある対象の型を表現出来ますが、それら
2597 はまた、概念の世界の実体を持たない対象を表現する事にも使えます(図 12.2 参照)

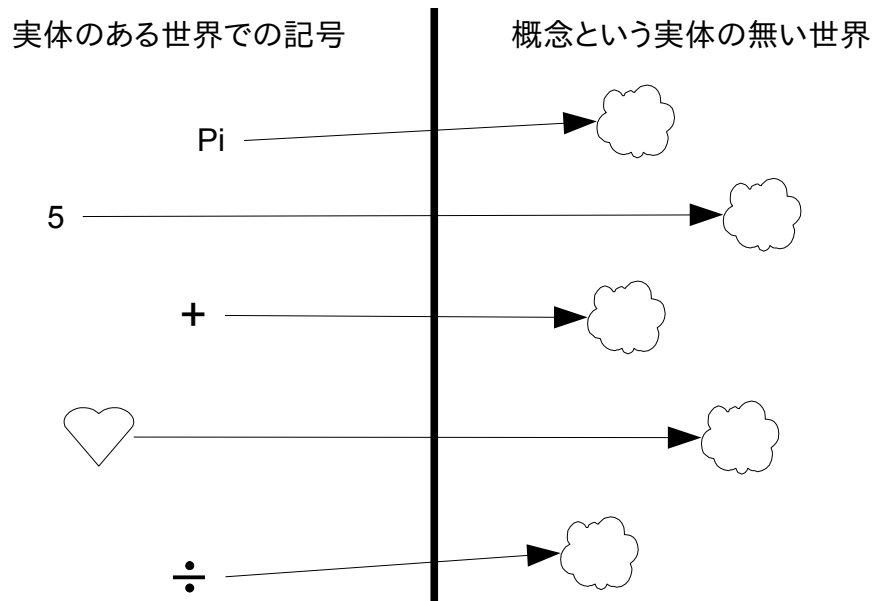


図 12.2: 実体のある記号は実体を持たない概念を表現出来る。

2598 実体を持ったものより構成された記号で最も単純なものの中はビット(0/1)とビット(0/1)の
2599 羅列です。ビット単体は **on** 状態と **off** 状態という二つの状態に単純に置き換えられます。
2600 **on** の状態を書いたり、印字したり、表示すると、数値 1 で表現され、**off** の状態は数値 0 で
2601 表現されます。ビットの並びを書いたり、印字したり、表示すると次の様なものになります:
2602 101, 100101101, 0101001100101, 10010.

2603 図 12.3 はビットの並びが、実体を持たない概念を表現する為、実体を持つ任意のから構成
2604 された記号として簡単に用いられる事を示すものです。

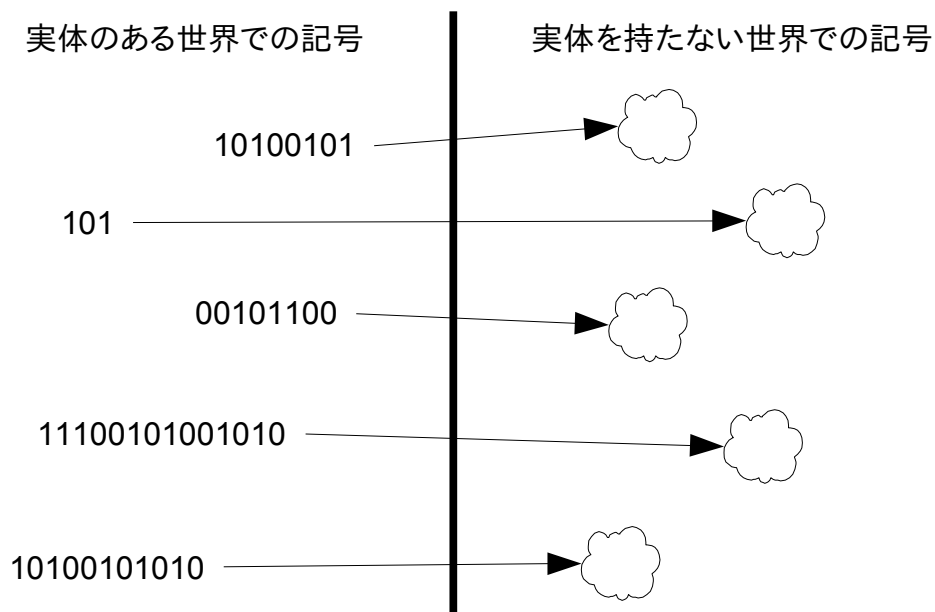


図 12.3: ビットは実体を持たない概念も表現出来る。

2605 実体を持ったものをビットやビットの並びとして構成する他の手法は:二つの周波数の間の音
2606 声信号のトーンを変えたり、灯りを点けたり消したり、ある物体の表面上に磁場を置いたり
2607 取り除いたり、電子デバイスで二つの水準の間で電圧を変えるとといった方法があります。
2608 大半の計算機が概念を表現するビットの並びを保つ為にさいごのしゅほうをもちいていま
2609 す。

2610 計算機の内部記憶は様々な”箱”、つまり、**記憶番地(memory location)**と各**記憶番地は**
2611 **概念を表現する為に用いることが可能なビットの並びで構成されています。**大半の計算
2612 機は数百万の番地をもっており、同時に数百万の概念を参照する事が容易に行えます。より
2613 大きな計算機は数十億の記憶番地を持っています。例えば、2007年版の典型的な個人向
2614 けの計算機は十億を越える記憶番地を持っています。

2615 図 12.4 に小さな計算機の内部記憶の断片にビットの並びが含まれている様子を示します。

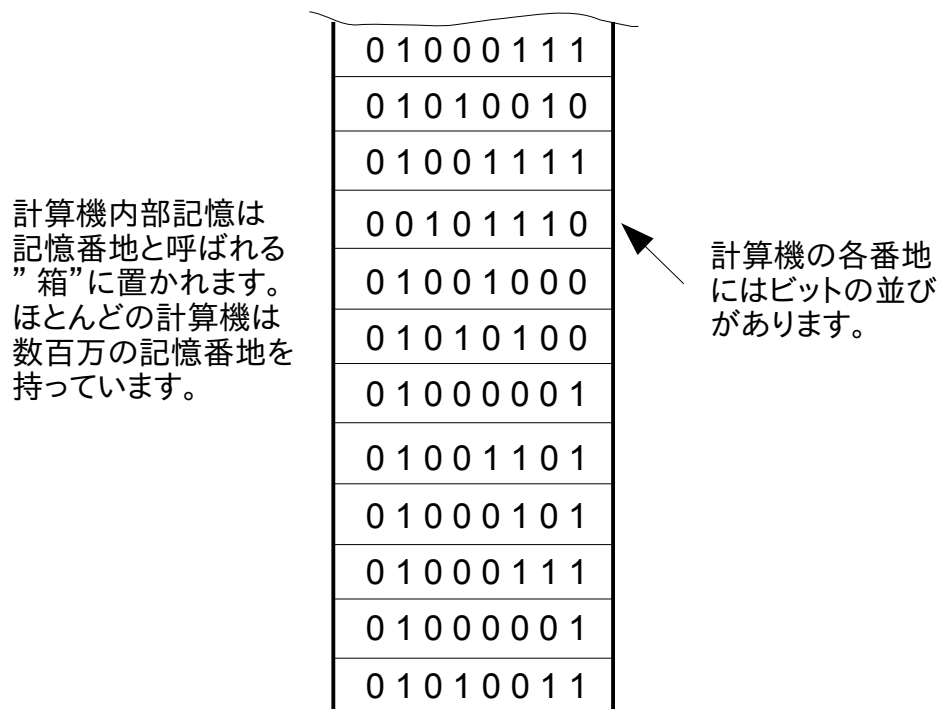


図 12.4: 計算機の記憶番地にはビットの並びがある。

2616 計算機内部記憶での何百ものビットの並びのそれぞれが、人間が思考し得る任意の概念を表
2617 現する能力を持っています。ほとんどの計算機が保有する莫大なビットの並びは、ある種の
2618 組織化された体系を使わず配置を保つことは困難です。

2619 計算機が保持する沢山のビットの並びを保つ為に用いている体系は、図 12.5 に示す様な一意
2620 な番地を各記憶番地に与えるものです。

01000111	11
01010010	10
01001111	9
00101110	8
01001000	7
01010100	6
01000001	5
01001101	4
01000101	3
01000111	2
01000001	1
01010011	0

各記憶番地は
計算機がそれを
指し示せる様に
一意に番地が
与えられています

図 12.5:各記憶番地には一意に番地が与えられます。

2621 12.4 文脈的意味

2622 この点で、貴方は” 記憶番地のビットの並び、すなわち、記憶番地の集まり、意味をどのよ
2623 うに決定しするの？” と不思議に思うかもしれません。この答えはビットの並びに意味を与
2624 える文脈的意味付けと呼ばれる概念になります。

2625 **文脈**は状況の中で生じた事象、あるいは何かが置かれた環境のことです。**文脈的意味付**
2626 は、すなわち、文脈中に置かれた事象や物事に対して与える意味付のことです。

2627 大半の人々が日々文脈的意味付を用いていますが、それに皆が気付いていません。文脈的意
2628 味付は非常に強力な概念であり、人間が思い浮かべられる任意の概念と計算機の記憶位置を
2629 結び付けるものです。もし、一つの記憶で与えられた並びを保つ為により多くのビットが必
2630 要であれば、その並びは一つの場所以上に広げられるのです。

2631 12.5 変項

2632 計算機は数を覚えておくことがとても得意なので、簡単に膨大な番地を覚えられます。人間
2633 は、残念なことに計算機の様にな数を覚える事は然程得意ではないので、この問題を解決する
2634 為に変項と呼ばれる概念が発明されました。 .

2635 変項は数の代わりに名前を使って記憶の中のビットの並び記号を人間が参照できる記憶番地
 2636 に結びつけられる名前です。 図 12.6 に計算機内部での4つの記憶番地に結びつけられた
 2637 四個の変項を示しておきます。

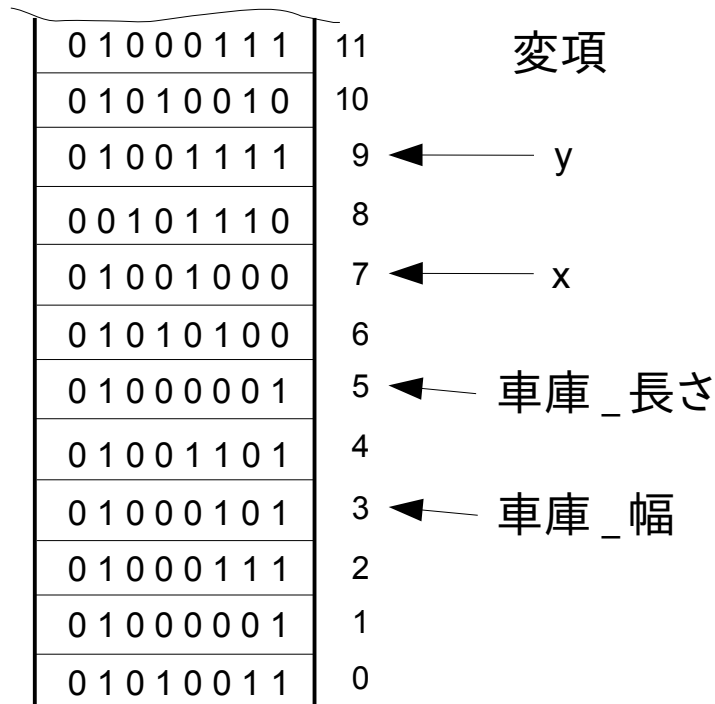


図 12.6: 記憶番地の代わりに変項を使うこと

2638 車庫_幅と車庫_長さと言う名前の変項は車庫の大きさを表現する並びが保存されている記
 2639 憶番地を参照し、変項 x と y は方程式に於ける数を表現するような記憶番地を参照していま
 2640 す。上の変項の記述が正確であるものの、これはさすがに使うには冗長で、人々が”変数
 2641 「車庫_長さ」は車庫の長さである”の様な発言や記述には時間がかかります。

2642 変数は記号的に対象の属性を表現する為に用いられます。典型的な個人向けの計算機でさえ
 2643 も数百万もの変数を擁する能力があるとはいえ、大半の対象が、ほとんどの計算機が擁する
 2644 ことのできる能力以上の莫大な属性を所有しています。例えば、1 キログラムの岩はおおよ
 2645 そ、10,000,000,000,000,000,000,000,000 個の原子を含んでいます。この岩の原子の位置
 2646 を表現することでさえも、最も進んだ計算機の能力を凌駕しているのです。それ故に、計算
 2647 機は通常、対象の完全な表現の代わりにモデルを用いて作業するのです。

2648 12.6 モデル

2649 モデルは対象の簡易化した表現で、対象のある種の属性のみに注目したものです。典型的な
 2650 対象の属性としては、重さ、長さや色彩が含まれます。モデルの為に選択される属性は所定

2651 の目的の為に選ばれます。モデルで表現される属性を増やせば増やす程、モデルの構築は高
2652 くつきます。それ故に、所定の目的を達成する上で絶対的に必要とされる属性のみがモデル
2653 で通常表現されるのです。対象の属性の幾つかだけをモデルの構成時に選択する行為を**抽象**
2654 **化**と呼びます。

2655 次の例はモデルを用いて問題を解く手順を描いたものです。2台の車が入る車庫を作ること
2656 を考えましょう。この車庫には車に沿って工作台、対の収納庫に芝刈り機があります。この
2657 倉庫は適切な天井の高さがあり、最初に述べた目的に必要な大きさ以上の車庫を作る気が無
2658 いと仮定すると、車庫の適切な長さ幅をどの様に決めれば良いでしょうか？

2659 車庫の大きさを決定するための一つの戦略は広い野原で大体 10 個のいろいろな大きさの車庫
2660 を作ることです。まず、車庫を作り終わると、二台の車を工作台、対の倉庫と芝刈り機と一
2661 緒に原っぱに置きます。それから、これらの物を車庫に交互に入れて、引っかからずに適合
2662 するもっとも小さな車庫を探します。

2663 原っぱのテスト車庫は破棄しても良く、選択した物と同じ大きさの車庫を希望する場所に建
2664 てられます。不幸にして、一つではなく 11 個の車庫がこの方式では必要で、これは非常に高
2665 価で非効率でしょう。

2666 この問題を費用をかけずに解く方法は**車庫のモデルとその中に置く物のモデル**を用いるこ
2667 とです。車庫の床の大きさを決めたいだけなので、紙を使って床の縮小モデルを作ることが
2668 出来ます。

2669 車庫に据える物も紙で縮小したもので表現する事が出来ます。すると、これらの物を表現す
2670 る紙製のものは床を表現する大きな紙の上に置く事ができ、紙の小物はどうすれば適合する
2671 かを見るために動かす事が出来ます。もし、これらの小物があまりにも窮屈であれば、床を
2672 表現する為に、より大きな紙を切れば良く、スカスカであれば、床の為により小さな紙を切
2673 れば良いのです。

2674 丁度良い具合になったら、床を表現する紙の長さや幅を計って、それらを実際の車庫向けに
2675 使える様に拡大します。この手法では、あとで捨ててしまう 10 個の実物の車庫の代わりに、
2676 幾らかの紙だけが問題を解くために必要とされます。

2677 紙の小物に複製された実物大の対象の属性は対象の長さや幅でした。この例で示す様に、紙
2678 のモデルはそれらが表現する対象よりも明らかに作業し易いものです。ところで、**計算機変**
2679 **数は紙、あるいは他の大半のモデル手法よりもモデル化で扱うにはより簡単なもので**
2680 **す。**

2681 この点で、紙モデル化の技術は我々が注目している計算機変数について一つの重要な長所が
2682 あります。紙モデルは小物のモデルを動かしたり、紙の車庫の床の大きさを変更すること
2683 によって変更可能です。我々が議論する変数は対象属性を表現する為に与えられた能力があり
2684 ますが、変数を変更する為の与えられた機構はありません。その変数の内容を変更する能力
2685 を持たない計算機は現実的に利用価値がありません。

2686 12.7 機械語

2687 計算機の記憶番地のビットの並びが人間が思考出来る任意の概念を表現する為に使えると以
2688 前述べました。記憶番地が任意の概念を表現していれば、それらが概念を参照できる事を意
2689 味し、この参照は計算機が自律的に記憶の中の変数をどの様に操作するか計算機に指図する
2690 命令(instruction)になります。

2691 記憶の中の命令に追随する計算機の側は中央演算装置(CPU)、すなわち、マイクロプロセッサ
2692 と呼ばれます。マイクロプロセッサが記憶の中の命令を追随しているときに、それらを走ら
2693 せているとか実行しているとも呼びます。

2694 マイクロプロセッサはファミリーに分類され、各マイクロプロセッサのファミリーはそれ自
2695 身の命令の集合(命令集合:instruction set と呼ばれます)を持っており、他のマイクロ
2696 プロセッサのファミリーが用いている命令とは異なっています。マイクロプロセッサの命令
2697 集合はマイクロプロセッサに何をさせるかを伝えることが出来る組み上げた積木の様な言語
2698 を表現します。この言語は記憶に送り込まれる命令集合からの命令列で構成されており、マ
2699 イクロプロセッサが理解できる唯一の言語です。これはマイクロプロセッサが理解できる唯
2700 一の言語なので、**機械語**と呼ばれます。計算機にやらせることを伝える為の機械語の命令の
2701 列は**計算機プログラム**と呼ばれ、計算機にやらせることを伝えるための機械語の命令の列
2702 を作成する人は**プログラマー**と呼ばれます。

2703 そこで、単純なマイクロプロセッサの命令集合がどのようなもので、この命令集合を使って
2704 構築された単純なプログラムを見てみましょう。

2705 これがマイクロプロセッサ 6500 ファミリーの命令集合です：

2706 ADC キャリー付きで記憶を加算器に追加(ADD memory to accumulator with Carry)。
2707 AND 記憶と加算器のANDをとる(AND memory with accumulator)。
2708 ASL 1ビット算術的にずらす(Arithmetic Shift Left one bit)。
2709 BCC キャリークリアなら分岐(Branch on Carry Clear)。
2710 BCS キャリーセットなら分岐(Branch on Carry Set)。

- 2711 BEQ 結果が零に等しければ分岐(Branch on result Equal to zero)。
- 2712 BIT 加算器のビットをメモリと比較(test BITs in accumulator with memory)。
- 2713 BMI 結果が負になれば分岐(Branch on result Minus)。
- 2714 BNE 結果が零に等しくなければ分岐(Branch on result Not Equal to zero)。
- 2715 BPL 結果が正なら分岐(Branch on result Plus)。
- 2716 BRK force Break.
- 2717 BVC オーバーフローフラグがクリアであれば分岐(Branch on oVerflow flag Clear)。
- 2718 BVS オーバーフローフラグがセットされれば分岐(Branch on oVerflow flag Set)。
- 2719 CLC キャリーフラグのクリア(Clear Carry flag)。
- 2720 CLD 十進モードをクリア(Clear Decimal mode)。
- 2721 CLI インタラプトディセイブルフラグをクリア(Clear Interrupt disable flag)。
- 2722 CLV オーバーフローフラグをクリア(Clear oVerflow flag)。
- 2723 CMP 記憶と加算器を比較(CoMpare memory and accumulator)。
- 2724 CPX 記憶と X レジスタを比較(CoMpare memory and index X)。
- 2725 CPY 記憶と Y レジスタを比較(CoMpare memory and index Y)。
- 2726 DEC 記憶を 1 減算(DECrement memory by one)。
- 2727 DEX レジスタ S を 1 減算(DECrement register S by one)。
- 2728 DEY レジスタ Y を 1 減算(DECrement register Y by one)。
- 2729 EOR 記憶と加算器の XOR を取る(Exclusive OR memory with accumulator)。
- 2730 INC 記憶に 1 加算(INCrement memory by one)。
- 2731 INX レジスタ X に 1 加算(INCrement register X by one)。
- 2732 INY レジスタ Y に 1 加算(INCrement register Y by one)。
- 2733 JMP 新しい記憶位置に跳ぶ(JuMP to new memory location)。
- 2734 JSR 新しいサブルーチンに跳ぶ(Jump to SubRoutine)。
- 2735 LDA 記憶から加算器に読み込み(Load Accumulator from memory)。
- 2736 LDX 記憶から X レジスタに読み込み(Load X register from memory)。
- 2737 LDY 記憶から Y レジスタに読み込み(Load Y register from memory)。
- 2738 LSR 論理的 1 ビット右にずらし(Logical Shift Right one bit)。
- 2739 NOP 無操作(No Operation)。
- 2740 ORA 記憶と加算器の OR を取る(OR memory with Accumulator)。
- 2741 PHA スタックに加算器のものを押し出す(PusH Accumulator on stack)。
- 2742 PHP スタックにプロセッサ状態を押し出す(PusH Processor status on stack)。
- 2743 PLA スタックから加算器に取り込み(PuLl Accumulator from stack)。
- 2744 PLP スタックからプロセッサ状態に取り込み(PuLl Processor status from stack)。
- 2745 ROL 1 ビット左にずらしてビット列を回します(ROtate Left one bit)。
- 2746 ROR 1 ビット右にずらしてビット列を回します(ROtate Right one bit)。
- 2747 RTI インタラプトから復帰(ReTurn from Interrupt)。

2748 RTS サブルーチンから復帰(ReTurn from Subroutine)。
 2749 SBC キャリーで減算(SuBtract with Carry)。
 2750 SEC キャリーフラグのセット(SEt Carry flag)。
 2751 SED 十進モードのセット(SEt Decimal mode)。
 2752 SEI インタラプトディセイブルフラグのセット(SEt Interrupt disable flag)。
 2753 STA 加算機の内容を記憶に保全(STore Accumulator in memory)。
 2754 STX Xレジスタの内容を記憶に保全(STore Register X in memory)。
 2755 STY Yレジスタの内容を記憶に保全(STore Register Y in memory)。
 2756 TAX 加算機の内容をレジスタ X に転送(Transfer Accumulator to register X)。
 2757 TAY 加算機の内容をレジスタ Y に転送(Transfer Accumulator to register Y)。
 2758 TSX スタックポインタの内容を X レジスタに転送(Transfer Stack pointer to register X)。
 2759 TXA Xレジスタを加算器に転送(Transfer register X to Accumulator)。
 2760 TXS Xレジスタをスタックポインタに転送(Transfer register X to Stack pointer)。
 2761 TYA Yレジスタを加算器に転送(Transfer register Y to Accumulator)。

2762 次は 6500 ファミリーの命令集合を使って書かれた小さなプログラムです。プログラムの目的
 2763 は 16 進数の番地 0200 から始まる記憶に置かれた 10 個の数の和を計算する事です。

2764 ここで、記憶の中の 10 個の数(これらは青で表示しています)が記憶位置にあり、そこに総和
 2765 が保管されます(これは赤字で表示されています)。ここで 0200 は記憶の中の最初の数の番地
 2766 です。

2767 0200 01 02 03 04 05 06 07 08 - 09 0A 00 00 00 00 00
 2768

2769 これらの 10 個の数の和を計算するプログラムです :

```
2770 0250 A2 00 LDX #00h
2771 0252 A9 00 LDA #00h
2772 0254 18 CLC
2773 0255 7D 00 02 ADC 0200h,X
2774 0258 E8 INX
2775 0259 E0 0A CPX #0Ah
2776 025B D0 F8 BNE 0255h
2777 025D 8D 0A 02 STA 020Ah
2778 0260 00 BRK
2779 ...
```

2780 このプログラムが実行されたあとに、それが計算した和は記憶に蓄えられます。和は 16 進数
 2781 で 37(十進数なら 55)になり、ここでは赤で示しておきます :

2782 0200 01 02 03 04 05 06 07 08 - 09 0A 37 00 00 00 00 007.....

2783 もちろん、貴方にこのアセンブリ言語のプログラムがどの様に働くかを理解することを期待
2784 していません。それを貴方に示した目的は、この様なマイクロプロセッサの命令集合を
2785 使ったプログラムがどの様なものであるかを貴方に見て頂く為です。

2786 低水準言語と高水準言語

2787 プログラマはその命令集合の命令を使って計算機のプログラムが出来ると入っても、これは
2788 うんざりする作業です。初期の計算機プログラムはマイクロプロセッサが理解できる機械語
2789 よりも自然言語、例えば英語にもっと似た言語でプログラムを開発することを望んでいまし
2790 た。機械語は**低水準言語**として考えられています。何故なら、それはマイクロプロセッサの
2791 回路で簡単に実行できる様に単純に設計されているからです。

2792 プログラマはそこでプログラムで彼らが使いたい**高水準言語**を低水準言語を使って作り上げ
2793 る事を思いました。こうして、FORTRAN (1957年)、ALGOL (1958年)、LISP (1959
2794 年)、COBOL (1960年)、BASIC (1964年)とC (1972年)が作られました。不幸なことに、マ
2795 イクロプロセッサは機械語しか理解できないので、高水準言語で記述された全てのプログラ
2796 ムはマイクロプロセッサで実行出来る様に機械語に変換しなければなりません。

2797 与えられたプログラミング言語向けにプログラムでどの様に入力するかを指図する規則のこ
2798 とを**構文規則**と呼びます。もしも、プログラマがプログラムの記述時に言語の構文規則に従
2799 わなければ、ソースコードを機械語に変換するソフトウェアが混乱し、所謂、構文エラーを
2800 返すことになります。

2801 構文エラーがどの様な物であることを示す例として、'印刷する'と云う言葉を考えてくださ
2802 い。もし、'印刷する'という言葉が所与のプログラム言語の命令であったとしましょう。そ
2803 こでプログラマが'印刷する'の代わりに'印刷する'と打ち込んでしまうと、これが構文エ
2804 ラーになるわけです。

2805 12.8 コンパイラとインタプリタ

2806 高水準言語から機械語に変換する為に用いられるプログラムに通常二種類あります。最初の
2807 種類が**コンパイラ**と呼ばれ、高水準言語のソースコード(通常は入力された書式です)をそ
2808 の入力として機械語に変換します。ソースコードと同値な機械語が生成されたのちに、それ
2809 が計算機の記憶に読込まれて実行されます。コンパイルされたプログラムはまた記憶装置に
2810 も保存可能で、そうすることで必要な時なら何時でも計算機の記憶に読込む事が出来ます。

2811 第二の機械語へ高水準言語を変換することに通常用いられる第二のプログラムの種類はイン
2812 タプリタと呼ばれます。コンパイラがする様なソースコードを機械語に変換する代わりに、
2813 インタプリタはソースコードを読み取り(通常は一度に一行)、ソースコードの一行で遂行す
2814 べきとされる処理を決定し、それからこれらの処理を実行します。そして、その下にある
2815 ソースコードの次の行を見つけると、プログラムのこの次の行で望まれる事の処理を決定す
2816 ると、これらの処理を実行し、以下同様となります。

2817 幾千もの計算言語が1940年代から生成されていますが、2から3百程度の歴史的に重要
2818 な言語が現在あります。ここで歴史的に重要な計算機言語の一覧を掲げるウェブサイトのリ
2819 ンクがあります：http://en.wikipedia.org/wiki/Timeline_of_programming_languages

2820 12.9 アルゴリズム

2821 計算機プログラマは少なくとも一つのプログラム言語は知っている必要がありますが、プロ
2822 グラム間が問題を解く時には、より抽象的な計算機言語よりも抽象性ではより高度な水準で
2823 それを遂行します。

2824 問題が解かれたのちには、その解がプログラム言語として展開されているのです。それはプ
2825 ログラマがあたかも二人の人であるかのようです。最初の間は**問題を解く人**であり、第二
2826 の人は**プログラム作成者**です。

2827 単純な問題では、多くのプログラマが心の中でアルゴリズムを生成し、それらのアルゴリ
2828 ムを直接プログラム言語に展開します。この過程での問題を解く人とプログラム作成者との
2829 間を切り替えています。

2830 複雑なプログラムになると、流石に、この問題を解く局面とプログラム作成の局面がより分
2831 離します。与えられた問題を解くアルゴリズムはプログラム言語よりも意味を用いて開発さ
2832 れ、それは文書の中に記録されます。この文書は問題を解く人からプログラム言語への展開
2833 を行うプログラム作成者への橋渡しとなっています。

2834 ある問題に対して問題を解く人が最初に行うことは**分析**です。これは殊に重要な段階です。
2835 というのも、問題を分析しなければ、適切に解く事が出来ないからです。あるものを**分析**
2836 するということは、構成部品にバラバラに分解して、これらの部分についてそれ等がどのよう
2837 にして動くかを定める為に研究する事を意味します。有名な文句は'**分解して征服せよ**
2838 **よ**'で、だから、難しい問題を分析する時には問題全体を解くよりも、それぞれをより単純
2839 にして小さな問題にバラバラにしてしまいます。**問題を解く人**は、より単純な問題をそれぞ
2840 れ解くアルゴリズムを開発し、それから、これらの**アルゴリズム**を結合して、問題全体に対
2841 する回を構成するのです。

2842 アルゴリズム(「ある-ご-りずむ」と発音します)は命令の列であり、この命令は与えられた
2843 作業をどのように達成するかを記述したものです。これらの命令は(英語の様な)自然言語に
2844 よる記述、それらの図式の描画、プログラム言語の記述を含めた様々な方法で 表現されてい
2845 ます。

2846 アルゴリズムの概念は数学者が数学の問題、例えば、二つの数の和やそれ等の積と云った様
2847 なものを解く為に関発した様々な手続に由来します。

2848 アルゴリズムはより一般的な問題にも使う事が可能です。例えば、紙モデルを使った車庫の
2849 大きさの問題を解きたいと考えている人にとって、次のアルゴリズムに従うことが出来るも
2850 のです：

2851 1) 車庫に据え付ける小物の長さや幅を計って、これらの計測を記録します。

2852 2) 第一からの計測を 100 で割って、それから、元の小物のモデルとして使えるような大き
2853 さに合致する紙片を切り出す。

2854 3) 紙片を切り出します。ここで紙片は長さを一番大きな車のモデルの長さの 1.5 倍、幅を 3
2855 倍とし、車庫の床モデルとします。

2856 4) 車庫床のモデルに車庫の扉を配置し、鉛筆を使ってその位置に印を付け、その紙の区画と
2857 鉛筆の印の間にちょうど入るように車庫の床モデルの上に両方の車を置きます。

2858 5) 紙の床モデルの上の車が占有していない空いている場所に小物を置きます。

2859 6) 小物のモデルを、それら全てが、この大きさの車庫の中で適合する様に、この空いた場所
2860 でいろいろな位置に動かしてみます。

2861 7) 受け入れられるものであれば段階 10 に行く。

2862 8) 車庫が窮屈だったら、車庫モデルの床の長さ、幅(あるいはその両方)を 10%増やして新し
2863 い床モデルを作って段階 4 に行く。

2864 9) もし、車庫に余裕が有り過ぎれば、車庫の長さ、幅(あるいはその双方)を 10%減らして、
2865 新しい車庫の床モデルを生成し、段階 4 に行きます。

2866 10)車庫の床モデルの長さと幅を計測し、これらを本来の大きさに拡大して戻し、これらの大
2867 きさの車庫を建てます。

2868 この例で見る事が出来る様に、アルゴリズムは往々にして、かなりの数の段階を含みます。
2869 何故なら、望ましい解を導出するに十分な程、詳細でなければならないからです。これらの
2870 段階を発展させて文書に記録したあとでやっと、その与えられた問題を解かなければならな
2871 い人々が、幾度も、その手順に従うことになるのです。

2872 12.1 計算

2873 人間がどのようにしてアルゴリズムの各段階に従うかを理解することはわりと容易な事です
2874 が、計算機のマイクロプロセッサが単純な機械言語の命令だけしか処理できない時に、これ
2875 らの段階をどのように計算機が処理をするのかを理解することはより難しいことです。

2876 マイクロプロセッサがアルゴリズムの各段階をどのようにして処理が出来ているかを理解す
2877 る為には、**計算**(これは**演算**としても知られています)とは何であるかを最初に理解しなけれ
2878 ばなりません。インターネット上でこれらの言葉のよい定義を探し、どう言っているかを読
2879 んでみましょう。

2880 **計算(computation)**という言葉には二つの定義があります：

2881 1) 固定された規則に従った数や記号の操作。通常、算術的電子計算機の操作に対して使われるが、心や脳で処
2882 理されたある種の過程の実行にも使われる。

2883 (www.informatics.susx.ac.uk/books/computers-and-thought/gloss/nodel.html)

2884 2) 計算は計算機と呼ばれる閉じた物理系内部で生じる純粋な物理的現象として観察できる。このような物理系
2885 の例としては、デジタル計算機、量子計算機、DNA 計算機、分子計算機、アナログ計算機や、wetware 計算機が
2886 ある(www.informatics.susx.ac.uk/books/computers-and-thought/gloss/nodel.html)

2887 これらの二つの定義は”**計算が決まった規則に基づいた数や記号の操作**”であり、”**計
2888 算機と呼ばれる閉じた物理的系内部での物理的現象として観察される**”ということを目指
2889 しています。双方の定義は、我々が通常計算機と呼んでいる機械が**計算機**という一つの種
2890 族であり、閉じた物理系の他の種族もまた計算機として動作することが出来ることを意味し
2891 ています。その他の計算機というものにはDNA 計算機、分子計算機、アナログ計算機や
2892 wetware 計算機(すなわち、脳)が含まれます。

2893 **演算(calculation)**の次の二つの定義は、通常の計算機、脳や他の種類の計算機を統べる
2894 ものに光を当てるものです：

2895 1)演算は一つ、あるいはそれ以上の入力を一つそれ以上の結果に変換する内的な手続である。
2896 (en.wikipedia.org/wiki/Calculation)

2897 2)演算：計算する手続き；数学的、あるいは論理的手法によって何かを決定すること
2898 (wordnet.princeton.edu/perl/webwn)

2899 演算に関するこれらの定義によると、“一つ、あるいはそれ以上の入力を一つ、あるい
2900 はそれ以上の結果に変換する内的な手続”であり、これは“数学的、あるいは論理
2901 的”に実行されるということを指しています。我々は脳が計算を実行する為に、どの様な数
2902 学的、論理的なものを用いるかをまだ完全に理解していませんが、この領域は急激に展開さ
2903 れている状態です。

2904 演算の第二の定義では**論理的**という言葉を使っており、話を進める前に、この言葉の定義が
2905 必要です：

2906 論理系は規則の全体を構成するもので、その体系の中での任意の意味付に用いられなければ
2907 なりません。数学の大半が良く理解された規則の構造上を根底とし、高度に論理的なもの
2908 として考えられています。任意の論理が適用される以前に、どの規則が用いられるかを述べた
2909 り、理解しておく必要が常にあります([ddi.cs.uni-](http://ddi.cs.uni-potsdam.de/Lehre/TuringLectures/MathNotions.htm)
2910 [potsdam.de/Lehre/TuringLectures/MathNotions.htm](http://ddi.cs.uni-potsdam.de/Lehre/TuringLectures/MathNotions.htm))

2911 **意味付(Reasoning)**はその体系での一つの点から別の点へ動かす為の予め定義された規則
2912 を用いる手続の事です。例えば、人が紙片の二つの数を足し合わせる時に、それらは正しい
2913 和を得る為の和のアルゴリズムの規則に従わなければなりません。和のアルゴリズム規則が
2914 その**論理**であり、誰かが演算の間に、これらの規則を適用する時、これらは規則によって**意**
2915 **味付**けられています。ここで、計算機がアルゴリズムの段階を、そのマイクロプロセッサ
2916 が単純な機械語命令が実行出来るだけの時に、どの様にして実行するかという疑問に対し
2917 て、これらの概念を適応してみましよう。

2918 人がアルゴリズムを開発するとき、アルゴリズムの段階は通常、各作業を実行する為に必要
2919 なより小さな段階の全てを含まない、高水準の作業として述べられています。

2920 例えば、ある人が“ニューヨークからサンフランシスコ迄の運転”という手引を書こうとしま
2921 す。この大きな手引は“その交差点を左に曲がり、西に10キロ進め等”の手順を含むより小さ
2922 なる手引に落とせます。大きな手引の全ての小さな手引が完成されれば、大きな手引も完成さ
2923 れます。

2924 この大きな運転の手引を使わなければならない人は通常、それを完遂する為に遂行しなければ
2925 ならない小さな手引を思い描く事が出来なければなりません。計算機は非常におバカなの
2926 で、任意のアルゴリズムは計算機上で実行出来る以前に、アルゴリズムの段階がより小さな
2927 段階に分けられて、それらの小さな段階がまたさらに小さな段階に分けられて、それらの段
2928 階がマイクロプロセッサの命令集合で実行されるのに十分に小さなものになっていなければ
2929 なりません。

2930 ある場合は、いくつかの小さな段階が大きな段階を実装する上で必要なだけかもしれませんが
2931 が、一方では数百、数千のより小さな段階が必要とされます。数百、数千の小さな段階は、
2932 アルゴリズムが機械語に変換される時に数百、数膳の機械語命令に変換されます。

2933 機械語は計算機でプログラム出来る唯一の言語であれば、人間によって計算機に送り込まれ
2934 るアルゴリズムの殆どが大き過ぎるものになるでしょう。高水準言語で記述されたアルゴリ
2935 ズムはしかしながら、機械語で必要とされるより小さくて沢山の段階に分割する必要がある
2936 ません。

2937 高水準言語で実装されるアルゴリズムを分解するという大変な作業は自律的にコンパイラや
2938 インタプリタで実行されます。これが時間の大半をプログラマが機械語の代わりに高水準言
2939 語を使って開発する理由です。

2940 12.2 アルゴリズムの記録に図式が使える

2941 前に、英語の様な自然言語でアルゴリズムは記録できるだけではないと述べましたが、図式
2942 を使って記録する事も出来ます。貴方は、殆ど図式に基づく言語が創られていて、プログラ
2943 ムが用いているアルゴリズムを含めて、'問題を解く人'がプログラムを設計出来ることを学
2944 べば驚くかもしれません。この言語はUMLと呼ばれ、Unified Modeling Language(統合
2945 モデル化言語)の事です。UML図式の一つは活動図式(activity diagram)と呼ばれ、幾
2946 つかの論理の組の段階(即ち、活動(activity))の列を示すことが出来ます。次の例は、アル
2947 ゴリズムが活動図式でどの様に表現されるかを示すものです。

12.3 1から10までの数の総和を計算

2948 分析や求解の前に問題で実行されなければならない第一のことは、明確に、そして明瞭に問
2949 題を記述することです。ここではアルゴリズムを使って求解する問題の短い陳述を挙げてお
2950 きます：

2951 **陳述：** この問題では、1から10の間に含まれる数の総和を決定することが必要とされる。

2952 これはかなり単純な問題なので、分析にたっぷり時間を使う必要はありません。 図 12.7 に
2953 活動図式に置き換えた、この問題を解くアルゴリズムを示しています。

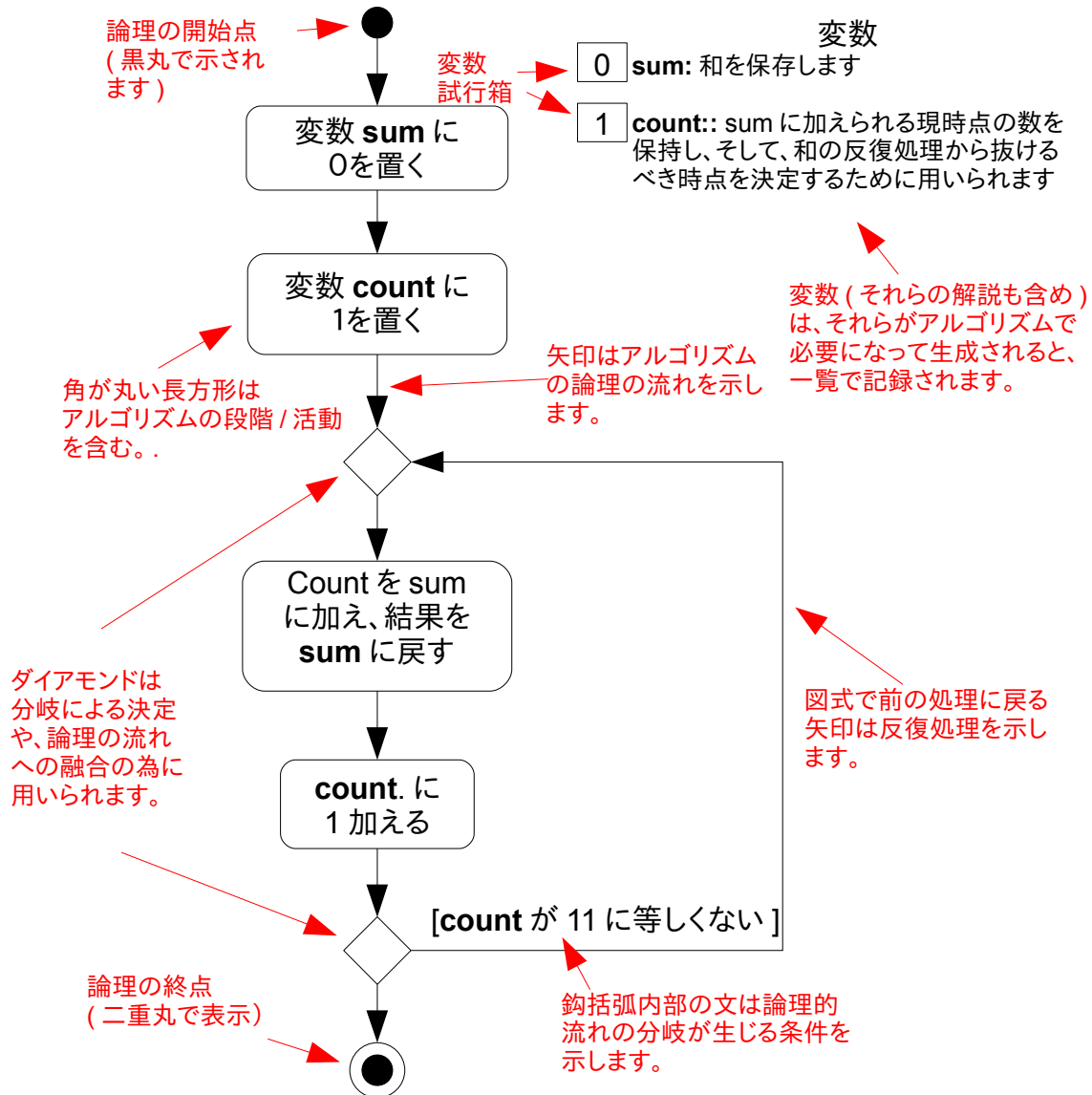


図 12.7: あるアルゴリズムの活動図式

2954 アルゴリズムと活動図式は同時に開発されます。開発途上、変数は必要に応じて生成され、
2955 それらの名前は通常、それらの陳述に沿ったリストに記録されます。開発者は定期的に出発
2956 点から開始して、それが正しい事を確認する為に論理的に一通り試してみます。
2957 試行箱は各変数の隣に置かれて、論理が変数をどのように変更するかを記録と更新ができる
2958 ようになっています。一通り試している間に、誤りが見つかったら、これらは流れの矢印を動かして、
2959 活動長方形の内部の文言を調整して修正しなければなりません。

2960 論理で間違いがないことを見つけると、開発者は問題を解く人でありつづける事を止めて、
2961 アルゴリズムをプログラミング言語でプログラムするプログラマに引き渡す事が可能になり
2962 ます。

2963 12.4 数学計算系の数学の部分

2964 数学は”並びの科学”とし記述されます。ここで並びの定義ですが：

2965 1) 体系的整理...

2966 (<http://www.answers.com/topic/pattern>)

2967 そして、系の定義は：

2968 1) 相互作用し、相互に関連し、あるいは互いに依存しあい、複雑な全体を構成する要素の一群。

2969 2) 相互の関連する概念や原理の体系化された集合。

2970 (<http://www.answers.com/topic/system>)

2971 それ故に、数学は物理的と非物理的対象の組織的な属性を扱う科学として考えられます。
2972 数学は非常に強力なので、全ての物理的、及び非物理的対象は組織的な属性を有し、それゆ
2973 えに数学は、理解されて操作され得るこれらの対象によって、ひとつの平均値となります。
2974 数学を知れば知るほど物理的世界に対して制御できるようになります。このことが数学を
2975 もっとも有用なものの一つで人間が得ることの出来る知識の中で興味深い領域にしているの
2976 です。

2977 伝統的に、数学は、人手の計算による数学に必要な、ワクワクすることでもなければ複雑な
2978 アルゴリズムを散々学ぶことを要求しました。典型的な数学の教科書の50%以上の内容が手
2979 書に基づくアルゴリズムの教示に割かれ、その上、教科書を通じた勉強では、より高い割合
2980 でこれらのアルゴリズムを人手で操作することに費やされています。

2981 ほとんどの人にとって、退屈な学習や実習、複雑な手計算アルゴリズムはあまりにも複雑
2982 で、心を踊らす事のない退屈なことで、決して数学のとてつもなく面白く、強力で、美しい
2983 側面を見る機会を持つことがないのです。

2984 悪いことに、手書中心の計算アルゴリズムは常にワクワクするものでもなく、複雑で退屈な
2985 ものです。良いことは数学計算環境の発見によって人々が手書中心のアルゴリズムを使う必
2986 要性が目立って減少した事です。

2987 13 SAGE サーバーの立ち上げ

2988 前節で示した様に、大半の人々が最初に SAGE をウェブサービスとして使い、この本の冒頭で
2989 読者は既に SAGE サーバーに接続したと思っても良いでしょう。

2990 この節は自分自身の SAGE サーバが欲しい人向けのもので、Windows や Linux 上での、入手、
2991 インストール、設定と保守を範囲とします。SAGE ノートブックサーバはインターネット技術
2992 に基づくので、この節はこれらの技術の幾つかを包含することで始めましょう。SAGE の構成
2993 の高水準な俯瞰は SAGE 配布ファイルに含まれている議論でつづけられるでしょう。最後
2994 に、Linux と Windows を基盤とする SAGE サーバー双方の立ち上げも包含します。

13.1 インターネットに基づく技術への入門

2995 インターネットは現在、我々の文明の最重要の技術の一つであり、その重要性は今後増大す
2996 る一方でしょう。実際、インターネットは非常に迅速に拡大しており、次の資料では、やが
2997 て殆ど全ての計算機器がそれに繋がる事を示しています
2998 (https://embeddedjava.dev.java.net/resources/waves_of_the_internet_telemetry.pdf)
2999 。

3000 だから、インターネットに関連する技術がどの様に働くのかを理解する事は計算機を使って
3001 仕事をする事に興味を持つ人々にとって価値のある事です。

3002 インターネットがどの様に創造されたかという歴史を理解する事も価値がありますが、我々
3003 はここで、この歴史を議論をするつもりはありません。何故なら、よそできちんと文書とし
3004 てまとめられているからです。私は貴方がインターネットの歴史についてインターネットの
3005 検索を使って、貴方が見つけた記事を読む事を高く勧めます。そうする事が貴方の時間の優
3006 れた投資となる事を私が保証します。

13.1.1 複数の計算機はどの様にして互いに通信するの？

3007 二つの計算機だけが互いに通信する必要がある時、状況は単純です。何故なら、必要とされ
3008 る全ての事は通信媒体（電線、光ファイバーケーブルや無線信号といった代物）で互いをつ
3009 なげる事です。一方の計算機から出発した情報はもう一つの計算機に送り込まれる云々で
3010 す。しかし、複数の計算機が互いに通信しなければならない状況であればどうでしょうか？
3011 この問題を解く方法は沢山ありますが、最も一般的な方法が図 1 1 で示す方法です：

3012 図 11 には、複数の計算機が所謂、局所ネットワーク、即ち、LAN と呼ばれるもので繋がっ
3013 ている様子を示しています。LAN は互いに物理的に閉じた複数の計算機（通常は同じ部屋や
3014 同じ建物の中）で構成され、通信媒介として用いられる物で互いに結合されています。図

3015 13.1 では計算機は銅線のイサーケーブルを使って switch と呼ばれる機器に繋がられていま
3016 す。

Local Area Network (LAN)

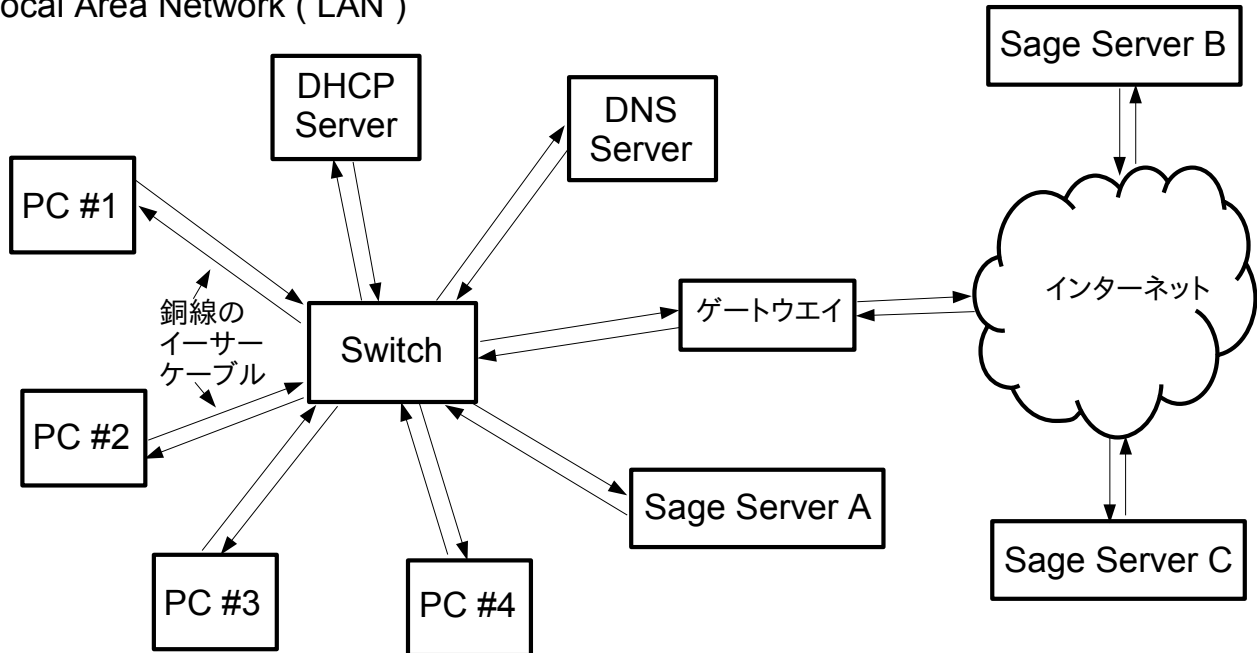


図 13.1: 局所ネットワーク(LAN)

3017 ネットワーク上の計算機は伝言を用いて互いに通信しますが、伝言を送る事は文通の手紙に
3018 似ています。switch の目的はそれに送り込まれる各伝言を監視し、その伝言が送り込まれ
3019 るべき計算機を決定し、それからその伝言を当の計算機に送り込みます。

3020 図 11 のモデルには問題があります。というのは、このネットワーク上の各計算機に結びつけ
3021 られた名前がそれらをユニークに同定することに、計算機の数が増加してしま
3022 えば不向きになるからです。これはさておき、絵の右側の雲はインターネットとそれに接続
3023 されている数百万の計算機(これらはホストとも呼ばれます)を表現しています。インター
3024 ネット上の各計算機がある方法でユニークに同定されれば、伝言はまたこれらの計算機に送
3025 られたり、これらから受け取られたりする事が出来ます。これはさておき、伝言がどの様に
3026 して交換されるべきかという規則も存在していなければなりません。

13.1.2 TCP/IP プロトコルについて

3027 インターネットが出現する以前、二つの解決すべき問題は 1)各計算機をユニークに同定する
3028 必要があったことと 2)どのようにメッセージが交換されるかを取り決める通信規則(プロト
3029 コルとも呼ばれます)を発達させる必要でした。インターネットのお陰で、プロトコルは

3030 「系の中の通信の為にキチンとした書式を定義する規則の集合」 (
 3031 www.unitedyellowpages.com/internet/terminology.html) として定義することが
 3032 できます。プロトコルの数は互いに用いられるときに、それらは**プロトコル集(protocol**
 3033 **suite)**と呼ばれます

3034 インターネット向けに発達したプロトコル集はTCP/IPと呼ばれ、その名前は、この集まりの
 3035 中で最も煩雑に用いられる二つの名前の組み合わせです(TCPは**転送制御プロトコル**
 3036 **(Transmission Control Protocol)**で、IPは**Internet Protocol**によります)。イ
 3037 ンターネットプロトコルは番地系を用いてインターネット上の計算機をユニークに同定す
 3038 る方法を定義しています。IP version 4 (**IPv4**)、これは現在もっとも広く用いられている
 3039 IPプロトコルの版で、**ドットで区切った0から255の間の4個の数字**で構成されていま
 3040 す。IPアドレスの例には207.21.94.50、54.3.59.2と204.74.99.100があります。0.0.0.0
 3041 から255.255.255.255までも全てのIPv4アドレスが番地空間を生成し、ここには
 3042 4,294,967,296番地あります。

3043 IP version 6 (**IPv6**)はIPプロトコルの最新版で、その番地空間が保有する番地は
 3044 340,282,366,920,938,463,463,374,607,431,768,211,456番地です！IPv4からIPv5へのい
 3045 こうは始まっていますが、ゆっくりとしています。大半のインターネット上のホストは長
 3046 い間IPv4を使い続けており、それ故に、IPv4はこの文書でも使われています。

3047 図 13.2 には 図 13.1 で示されたネットワークと同じモデルを含んでいますが、各計算機は

IPv4 番地を使った局所ネットワーク (LAN)

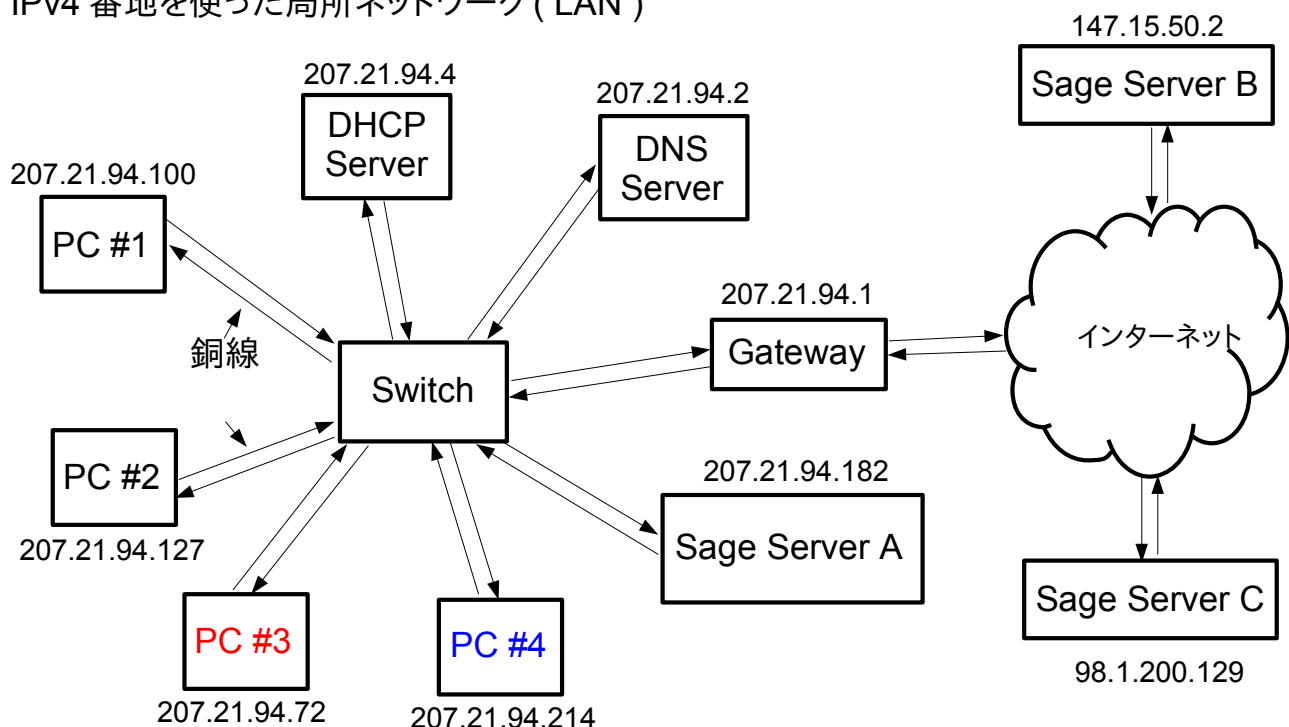


図 13.2: IP アドレス

3048 IPv4 番地が割り当てられています :

3049 もし、PC #3 が PC #4 に伝言を送る必要があれば、PC #4 の IP アドレス(これは
3050 207.21.94.214 です)に宛てて伝言を送り込みます。送り手の IP アドレス(207.21.94.72)も
3051 また、PC #4 が返事を送る必要があった場合に備えて、伝言の中に置かれます(これは返送先
3052 の番地を手紙に添えておくことと同様です)。PC #3 はそれから switch に伝言を送
3053 り、switch は伝言の送り先の番地を探し、それから PC #4 に伝言を引き渡します。

3054 もし、この局所ネットワーク上の一つの計算機が、その LAN 上にない計算機に伝言を送る必
3055 要があれば、その伝言はゲートウェイ計算機に送られて、そのゲートウェイはそれからその
3056 伝言をインターネットに流します。

13.1.3 クライアントとサーバー

3057 LAN 上やインターネット上で、組織化された計算機同士が通信する幾つかの方法があり、そ
3058 して、それらの組織の事を構成(**architecture**)と呼びます。一つの構成は **Peer-to-**
3059 **Peer** (P2P) とよばれ、ネットワーク上の計算機を互いに情報を交換する等価なものとして扱
3060 います。P2P アプリケーションの例はインスタントメッセージングです。

3061 ネットワーク上の計算機を使った別の構成は **クライアント-サーバ** と呼ばれます。クライア
3062 ント・サーバー構成では、サーバーはネットワーク上の他の計算機の要求を受け入れる計算
3063 機で、要求された仕事を果たし、請求者に向けて作業結果を返します。クライアントは計算
3064 機で、サーバーに要求を送って、応答を受け取って、それから応答に含まれている情報を利
3065 用します。

3066 図 11 に示す LAN では、三個のサーバ(DHCP サーバー、DNS サーバと SAGE サーバー)と四個の
3067 クライアントがあります。DHCP と DNS サーバーは次の二つの節で議論しましょう。

13.1.4 DHCP

3068 DHCP は **Dynamic Host Configuration Protocol** (動的ホスト設定プロトコル)の事で、
3069 その目的は LAN 上の計算機を立ち上げるときにネットワークの接続に必要な情報を使って自
3070 律的に設定させることです。この情報には IP アドレス、ゲートウェイのアドレスに DNS
3071 サーバーのアドレスが含まれます。IP アドレスやゲートウェイが何であるかを我々は既に
3072 議論しています。DNS サーバーは次の節で解説することにしましょう。

3073 IP アドレスを持っていない(何故なら立ち上げだから)計算機がどの様にして IP アドレス
3074 を取得する為に DHCP サーバーと接続する為にネットワークが使えるかという点に疑問を感じ

3075 るかもしれません。この問題は立ち上げ時の計算機が LAN に対して **broadcast** メッセージ
3076 を流すことで解決されます。broadcast メッセージは LAN 上の特定の計算機に送られるもの
3077 ではありません。その代わりに、broadcast メッセージは LAN に対して送られて、LAN 上の全
3078 ての計算機がその伝言を受け取ります。

3079 DHCP 要求伝言が LAN 上に送り込まれると、DHCP サーバーは他の計算機と同時に要求を受け取
3080 ります。その他の計算機は伝言の内容を読み、その DHCP 要求を含む伝言を見て、それを無視
3081 します。DHCP サーバーは、伝言の内容を読んで、その伝言がそれに対して意味することを理
3082 解すると、DHCP 設定情報を送り主に送り返すのです。

13.1.5 DNS

3083 インターネット上の数百万もの計算機の各々がそれらの IP アドレスを使ってアクセスされま
3084 す。例えば、sagemath.org ウェブサイトを包含するサーバーの IP アドレスは
3085 **12.208.160.192** です。ウェブブラウザを立ち上げて、**http://128.208.160.192/sage**
3086 と入力すれば、貴方は直接このウェブサイトへ接続できます。

3087 莫大な数を覚えることは人間にとっては困難な事ですが、しかしながら、**IP アドレス番号**
3088 **を使って関連付けた名前**で構成された系がインターネット向けに生成されています。その
3089 系の名前が **DNS** で、**Domain Name System** のことです。一つ以上の IP アドレスに対応付け
3090 られた名前は **ドメイン名 (domain name)** と呼ばれ、頭に与えられた計算機の **ホスト名** (と
3091 お尻にピリオド)があるドメイン名のことを **fully qualified ドメイン名** と呼びます。
3092 ドメイン名の例としては：

3093 gentoo.org
3094 yahoo.com
3095 sourceforge.net
3096 google.com
3097 sagemath.org
3098 wikipedia.com

3099 fully qualified ドメイン名の例は：

3100 kiwi.gentoo.org.
3101 loon.gentoo.org.
3102 wren.gentoo.org.

3103 DNS はインターネット全体に渡って分散した膨大なデータベースとして実装されています。

3104 ドメイン名は、それらを DNS 系に入れる前に、**ドメイン名登録**機関で登録しておく必要があ
3105 ります。ドメイン名登録会社の例には godaddy.com、networksolutions.com と
3106 register.com があります。

3107 図 12 の LAN 上の DNS サーバーは三つの機能を持っています。最初の機能はドメイン名を含む
3108 クライアントからのメッセージを受理し、それらの名前に対応する IP アドレスを返すことで
3109 す。利用者が sagemath.org の様なドメイン名をブラウザの URL バーに入力したときに、ブラ
3110 ウェーはまだ SAGE ウェブサイトと接触出来ません。というのも、その IP アドレスを知らない
3111 からです。ブラウザが動作しているオペレーティングシステムは、そのために、ドメイン名
3112 を (DHCP を通じて得た DNS サーバーの IP アドレスを使って) DNS サーバーに送りつけます。す
3113 ると、DNS サーバーは sage.org ドメイン名に関連する一つ以上の IP アドレスを返します。そ
3114 こで、システムは SAGE サーバーが乗っているサーバーと背食するために、これらの IP アド
3115 レスを使います。

3116 局所 DNS サーバが持つ第二の機能は局所ネットワーク上の計算機に対応する IP アドレスに
3117 対して**ドメイン名**を定義することです。もし、インターネット上で遠くにある計算機がその
3118 局所ネットワーク上のある計算機の IP アドレスを知りたがっているものの、DNS サーバがそ
3119 の対応を知らないとします。遠隔 DNS サーバが局所**権威(authotative)**DNS サーバに接触し
3120 て、その対応がどうなっているかを尋ねます。そして、遠隔 DNS サーバは一定期間、今後、
3121 その遠隔ネットワーク上の計算機が対応関係を知る必要がある場合に備えて、この対応関係
3122 を覚えておきます。

3123 第三の DNS サーバが持つ機能は IP アドレスを含む伝言を取得し、これらのアドレスに対応
3124 する**ドメイン名**に返却することです。

13.1.6 プロセスとポート

3125 ここで、インターネットに関連するより重要な幾つかの技術について議論してきました
3126 が、IP メッセージ(今後、メッセージとして参照します)が計算機に届いたときに何が生じ、
3127 ある計算機から送られたメッセージの前に何がメッセージを生成するかを話す時です。

3128 殆ど全ての個人向けの計算機では同時に計算喜寿尾で複数のプログラムを動かさせます。ここ
3129 では典型的な利用者の計算機上で同時に動作しているプログラムのリストを挙げておきま
3130 しょう：

- 3131 - ウェブブラウザ
- 3132 - インスタントメッセージクライアント
- 3133 - ワープロ

- 3134 - ファイルダウンロードユーティリティ
- 3135 - 音楽ファイルプレイヤー
- 3136 - ゲーム

3137 殆どの計算機のオペレーティングシステムで、動作しているプログラムは**プロセス**と呼ばま
3138 す。Windows では現時点で動作している全てのプロセスの一覧は<ctrl><alt>と<delete>キー
3139 を同時に押すことで起動するタスクマネージャーで見られます。Linux の様な UNIX 系のシ
3140 ステムでは、動作しているプロセスの一覧は **ps -e** 命令を実行することで得られます。ここ
3141 で、私が使っている Linux システム上で動作しているプロセスの一覧を示しておきましょ
3142 う：

```
3143 manage@sage:~$ ps -e
3144  PID TTY          TIME CMD
3145   1 ?            00:00:00 init
3146   2 ?            00:00:00 ksoftirqd/0
3147   3 ?            00:00:00 watchdog/0
3148   4 ?            00:00:00 events/0
3149   5 ?            00:00:00 khelper
3150   6 ?            00:00:00 kthread
3151   8 ?            00:00:00 kblockd/0
3152   9 ?            00:00:00 kacpid
3153  10 ?            00:00:00 kacpi_notify
3154  67 ?            00:00:00 kseriod
3155 100 ?            00:00:00 pdflush
3156 101 ?            00:00:00 pdflush
3157 102 ?            00:00:00 kswapd0
3158 103 ?            00:00:00 aio/0
3159 1545 ?            00:00:00 scsi_eh_0
3160 1547 ?            00:00:00 scsi_eh_1
3161 1728 ?            00:00:02 kjournald
3162 1796 ?            00:00:00 logd
3163 1914 ?            00:00:01 udevd
3164 2611 ?            00:00:00 shpchpd
3165 2620 ?            00:00:00 kpsmoused
3166 3208 tty2          00:00:00 getty
3167 3209 tty3          00:00:00 getty
3168 3210 tty4          00:00:00 getty
3169 3211 tty5          00:00:00 getty
3170 3212 tty6          00:00:00 getty
3171 3263 ?            00:00:00 dd
3172 3265 ?            00:00:00 klogd
3173 3345 ?            00:00:14 vmware-guestd
3174 3381 ?            00:00:00 sshd
3175 3404 ?            00:00:00 atd
```

```
3176 3414 ?          00:00:00 cron
3177 3959 ?          00:00:00 dhclient3
3178 4140 tty1        00:00:00 login
3179 4141 tty1        00:00:00 bash
3180 4429 ?          00:00:00 syslogd
3181 4507 ?          00:00:00 sshd
3182 4508 pts/1        00:00:00 bash
3183 4538 tty1        00:00:00 sage
3184 4541 tty1        00:00:00 sage-sage
3185 4554 tty1        00:00:00 python
3186 4555 tty1        00:00:05 sage-ipython
3187 4573 tty1        00:00:00 sh
3188 4574 tty1        00:00:00 sage
3189 4580 tty1        00:00:00 sage-sage
3190 4591 tty1        00:00:02 python
3191 4592 pts/2        00:00:00 sage
3192 4600 pts/2        00:00:00 sage-sage
3193 4611 pts/2        00:00:06 python
3194 4611 pts/1        00:00:00 ps
```

3195 もし、貴方がこのリストの下側を見れば、SAGE ノートブックサーバと一書に SAGE が動作し
3196 ている事が分かるでしょう。ps 命令はそれ自身を一覧に含めていますが、その一覧が生成さ
3197 れた時点で、それが動作していたからだということに注意してください。

3198 この一覧には四列あります。各プロセスには、プロセスが生成された時点で単一の**プロセス**
3199 **ID(PID)**番号が与えられ、これらの番号は**PID**列に挙げられています。**TTY**列はプロセスが
3200 端末に結びつけられているかどうかと、もしも繋がっていれば、どの端末に結び付いている
3201 かを示しています。**TIME**列はCPU時間をそのプロセスが時間、分と秒でどれだけ使ったかを
3202 示しています。

3203 ネットワークから計算機にメッセージが届いたときに、計算機はどのプロセスがメッセージ
3204 を受けとるかを判断しなければなりません。TCP/IP プロトコルはこの問題を解く手法で、ソ
3205 フトウェアに基づく**通信ポート**を使います。

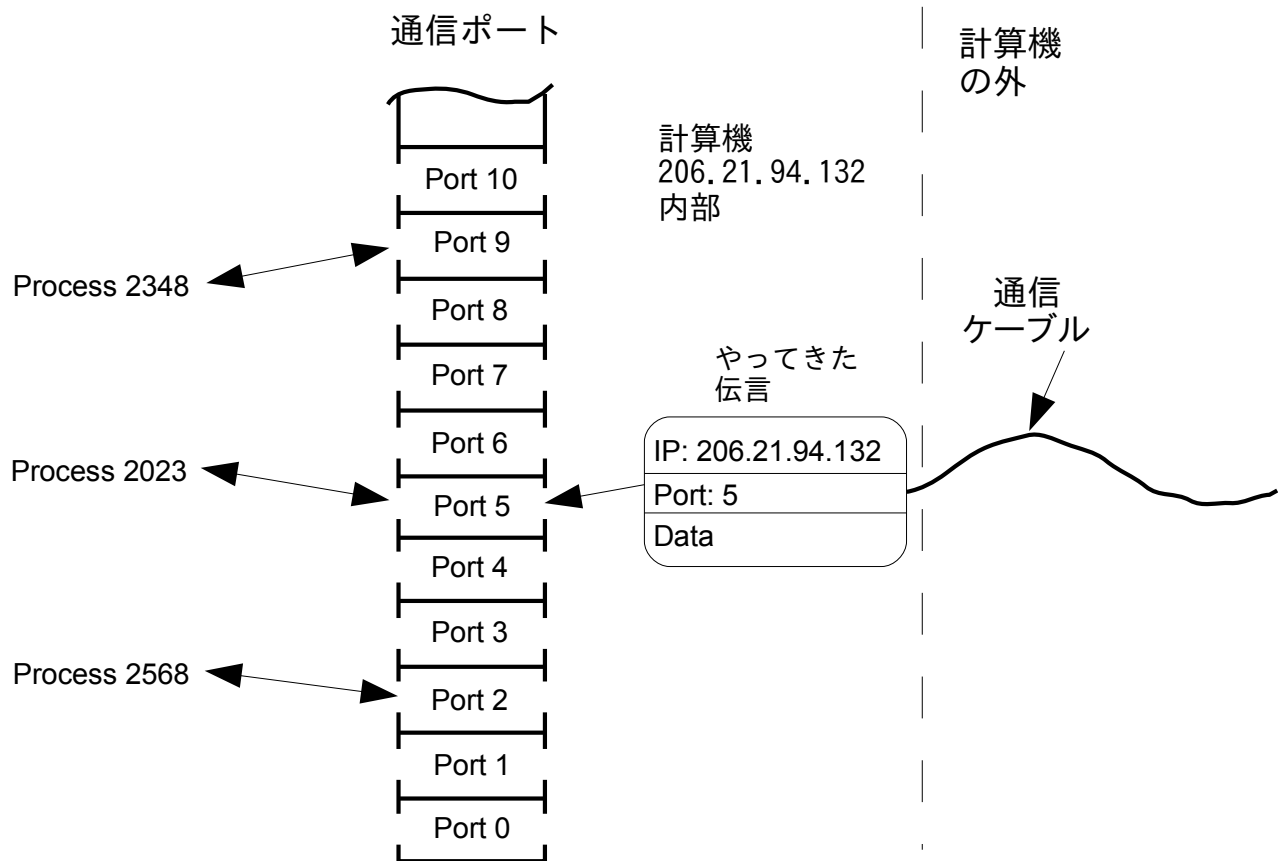


図 13.3: 通信ポート

3206 図 13.3 では、ネットワークに繋がった IP アドレスが 206.21.94.132 の計算機の中と外を示
 3207 しています。通信ポートはプロセス間に置かれ、動いているプロセスは左側、右側にはネット
 3208 ネットワーク接続があります。各ポートは最低のポート番号が 0 から最も高いポート番号が
 3209 65535 で単一の番号が与えられています。ネットワークから届く各伝言は、その中にポート
 3210 番号を持っており、その番号には系がどのポートが伝言が送ったかが分かるようになってい
 3211 ます。

3212 図 13.3 で、port 5 を行き先に持つ伝言がネットワークから到着しているので、系は port
 3213 5 にその伝言を送り込みます。process 2023 は系が port 5 に伝言を送り込んだ時点でそ
 3214 のポートに束縛され、process 2023 は伝言を受け取り、それからそれに含まれる情報を
 3215 使って何かを実行します。

3216 図 13.4 に IP アドレスが 65.2.8.3 を持つネットワーク上の別の計算機に送られた process
 3217 2023 からの伝言を示します。この伝言が目的の計算機に到達したときに、その伝言をその

3218 ポート 8 に置けば、port 8 に束縛されたその計算機のプロセスがある筈です。

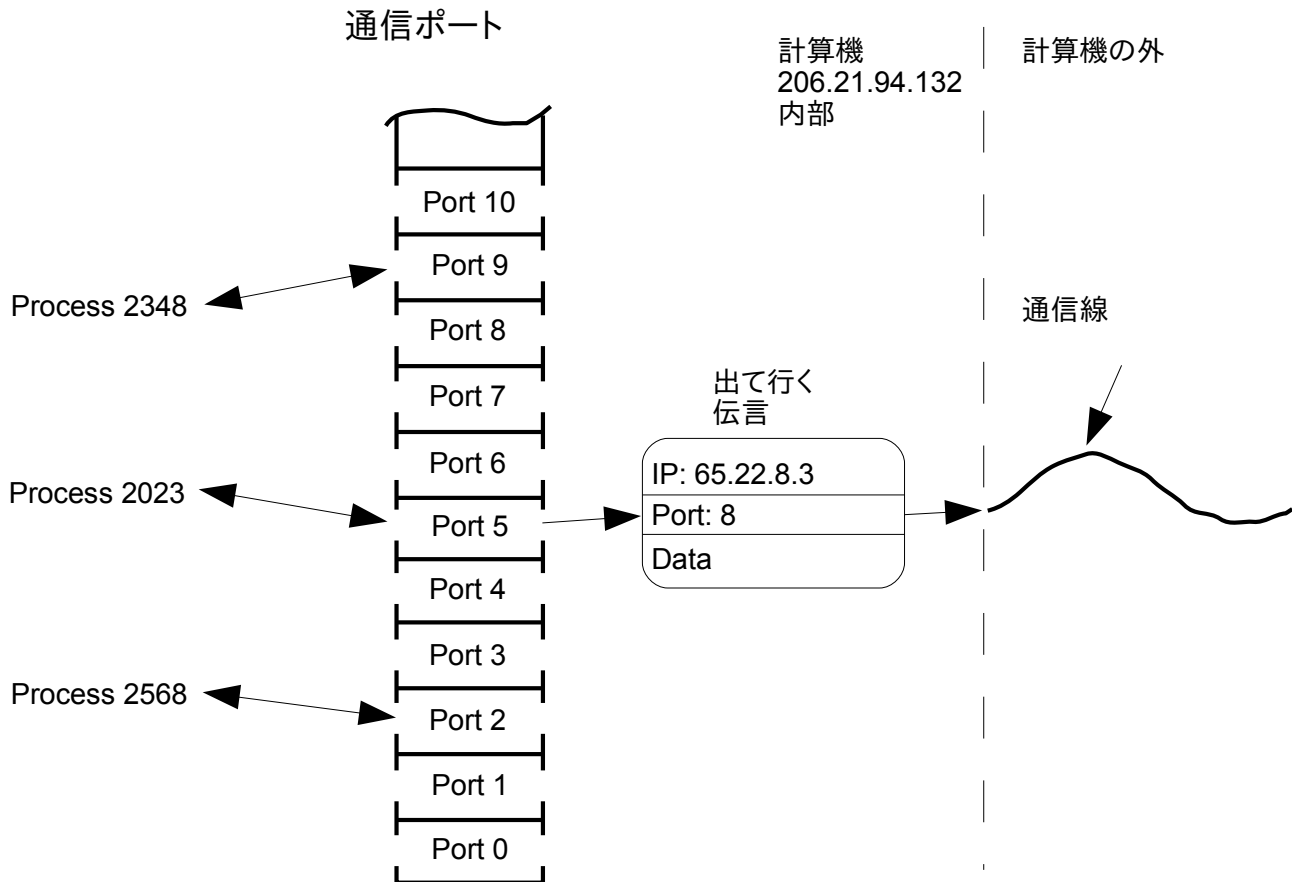


図 13.4: 出ていった伝言

13.1.7 良く知られたポート，登録されたポートと動的ポート，

3219 ここで、貴方はポートが何であり、それらに拘束されたプロセスがどうするかが分かっています。人がどのプロセスがどのポートに束縛されるべきかをどうやって決定するか不思議に思っているかもしれません。IANA(Internet Assigned Numbers Authority)と呼ばれる組織が、インターネットに関連する様々な手法、その一つがTCP/IPポート手法ですが、に関して責任を持っています。IANAは0 - 635535ポート範囲を次の三つの区画に分けています：

3224 0 - 1023 -> 著名なポート

3225 1024 - 49151 -> 登録されたポート

3226 49152 - 65535 -> 動的、或いは個人的ポート

3227 13.1.7.1 著名なポート (0 - 1023)

3228 IANA が管理している一覧には、どの種類のプログラムが、この範囲内の特定のポート番号に
 3229 通常、束縛されるかが示されています。例えば、ウェブサーバは port 80 に束縛さ
 3230 れ、SSH(secure shell)サーバは port 22 に束縛され、FTP(ファイル転送プロトコ
 3231 ル:File Transfer Protocol)サーバは port 20 に束縛されて、DNS サーバーは port
 3232 53 に束縛されています。ここで、最初の 25 の著名なポートを示しますが、一覧全体は
 3233 <http://www.iana.org/assignments/port-numbers> から入手できます:

3234	キーワード	十進数	概要	参照文献
3235	-----	-----	-----	-----
3236		0/tcp	Reserved	
3237		0/udp	Reserved	
3238	#		Jon Postel <postel@isi.edu>	
3239	tcpmux	1/tcp	TCP Port Service Multiplexer	
3240	tcpmux	1/udp	TCP Port Service Multiplexer	
3241	#		Mark Lottor <MKL@nisc.sri.com>	
3242	compressnet	2/tcp	Management Utility	
3243	compressnet	2/udp	Management Utility	
3244	compressnet	3/tcp	Compression Process	
3245	compressnet	3/udp	Compression Process	
3246	#		Bernie Volz <volz@cisco.com>	
3247	#	4/tcp	Unassigned	
3248	#	4/udp	Unassigned	
3249	rje	5/tcp	Remote Job Entry	
3250	rje	5/udp	Remote Job Entry	
3251	#		Jon Postel <postel@isi.edu>	
3252	#	6/tcp	Unassigned	
3253	#	6/udp	Unassigned	
3254	echo	7/tcp	Echo	
3255	echo	7/udp	Echo	
3256	#		Jon Postel <postel@isi.edu>	
3257	#	8/tcp	Unassigned	
3258	#	8/udp	Unassigned	
3259	discard	9/tcp	Discard	

3260	discard	9/udp	Discard
3261	#		Jon Postel <postel@isi.edu>
3262	discard	9/dccp	Discard SC:DISC
3263	#		IETF dccp WG, Eddie Kohler <kohler@cs.ucla.edu>, 3264 [RFC4340]
3265	#	10/tcp	Unassigned
3266	#	10/udp	Unassigned
3267	systat	11/tcp	Active Users
3268	systat	11/udp	Active Users
3269	#		Jon Postel <postel@isi.edu>
3270	#	12/tcp	Unassigned
3271	#	12/udp	Unassigned
3272	daytime	13/tcp	Daytime (RFC 867)
3273	daytime	13/udp	Daytime (RFC 867)
3274	#		Jon Postel <postel@isi.edu>
3275	#	14/tcp	Unassigned
3276	#	14/udp	Unassigned
3277	#	15/tcp	Unassigned [was netstat]
3278	#	15/udp	Unassigned
3279	#	16/tcp	Unassigned
3280	#	16/udp	Unassigned
3281	qotd	17/tcp	Quote of the Day
3282	qotd	17/udp	Quote of the Day
3283	#		Jon Postel <postel@isi.edu>
3284	mss	18/tcp	Message Send Protocol
3285	mss	18/udp	Message Send Protocol
3286	#		Rina Nethaniel <----none---->
3287	chargen	19/tcp	Character Generator
3288	chargen	19/udp	Character Generator
3289	ftp-data	20/tcp	File Transfer [Default Data]
3290	ftp-data	20/udp	File Transfer [Default Data]
3291	ftp	21/tcp	File Transfer [Control]
3292	ftp	21/udp	File Transfer [Control]
3293	#		Jon Postel <postel@isi.edu>
3294	ssh	22/tcp	SSH Remote Login Protocol
3295	ssh	22/udp	SSH Remote Login Protocol
3296	#		Tatu Ylonen <ylo@cs.hut.fi>

3297 telnet 23/tcp Telnet
3298 telnet 23/udp Telnet
3299 # Jon Postel <postel@isi.edu>
3300 24/tcp any private mail system
3301 24/udp any private mail system
3302 # Rick Adams <rick@UUNET.UU.NET>
3303 smtp 25/tcp Simple Mail Transfer
3304 smtp 25/udp Simple Mail Transfer

3305 ネットワーク上の一つの計算機がネットワーク上の別の計算機上の特定のサービスを利用し
3306 たいと希望したとき、最初の計算機は伝言を作成し、その伝言の中に希望するサービスの
3307 ポート番号に置いて、それから伝言を行き先の計算機に送ります。もし、そのポート向けの
3308 著名なサービスを行うプロセスがそのポートに束縛されていれば、このプロセスがその伝言
3309 を受け取り、要求された作業を実行します。

3310 著名なポート範囲内のポートに束縛されているプロセスについての主要な制約は、それらが
3311 特権利用者権限(super user priviledge)で動作していなければならないことです。

3312 13.1.7.2 登録されたポート (1024 - 49151)

3313 登録されたポートは著名なポートと同様に動作していますが、関連付けられたプロセスは
3314 特権利用者権限で動作している必要がありません。登録されたポートの一覧は、著名
3315 なポートの一覧を含んでいるのと同じ IANA 文書に含まれています。

3316 13.1.7.3 動的/個人ポート (49152 - 65535)

3317 これらのポートは必要時応じて使われ、それらに関連する特定のプロセス型はありません。
3318 この領域のポートの典型的な利用は、ウェブブラウザ向けにウェブサーバを用いた外部接続
3319 です。

13.1.8 SSH (Secure SHell) サービス

3320 著名なポートを経由して利用されるサービスの例として、SSH(Secure Shell)サービスがあ
3321 り、それは通常 port 22 に束縛されています。SSH サービスはネットワーク上の別の計算
3322 機から利用者がログインすることを許可します。遠隔計算機にログインする前に、利用者は
3323 利用者名とパスワードを知っていなければなりません。そして、遠隔計算機は(プロセスの
3324 形式で)動作し、port 22 に束縛された SSH サービスを持っていなければなりません
3325 。SSH は計算機間を、その間を行き来するデータの暗号化によって機密を守った接続が行え

3326 ます。

3327 UNIX系のシステムでは、SSHクライアントプログラムは単にSSHと呼ばれ、Windowsシステム
3328 では、貴方はsshサービスが動作している計算機にリモートでログインが行える様に
3329 `putty.exe` と呼ばれるプログラムをダウンロードとインストールを行わなければなりません。
3330 この `putty.exe` プログラムは(
3331 <http://www.chiark.greenend.org.uk/~sgtatham/putty/download.html>)から入手できます。

3332 `ssh` クライアントプログラムが最初にリモートマシンにログインしようとする時、利用者に
3333 このホストに対する暗号情報を現時点で持っていない事を告げて、続けるかどうかを尋ねま
3334 す。"yes" と入力することで答えます。すると、このプログラムはこのホストに関する情報
3335 を既知のホストの一覧に追加し、その後は二度と質問をしません。

13.1.9 ネットワーク上の計算機間のファイルを複写する為の `scp` の利用について

3336 SSHサービスは利用者が計算機にリモートログインを許容するだけではなく、ネットワーク
3337 上の計算機の間でのファイルの複写を行う為にも使えます。ファイル複写の為にLinuxのクラ
3338 イアントプログラムは `scp` (Secure Copy) と呼ばれ、そして、人気のあるWindows `scp` クラ
3339 イアントの `pscp.exe` は `putty.exe` と同じurlから入手出来ます。

3340 13.2 SAGE の構成 (まだ)

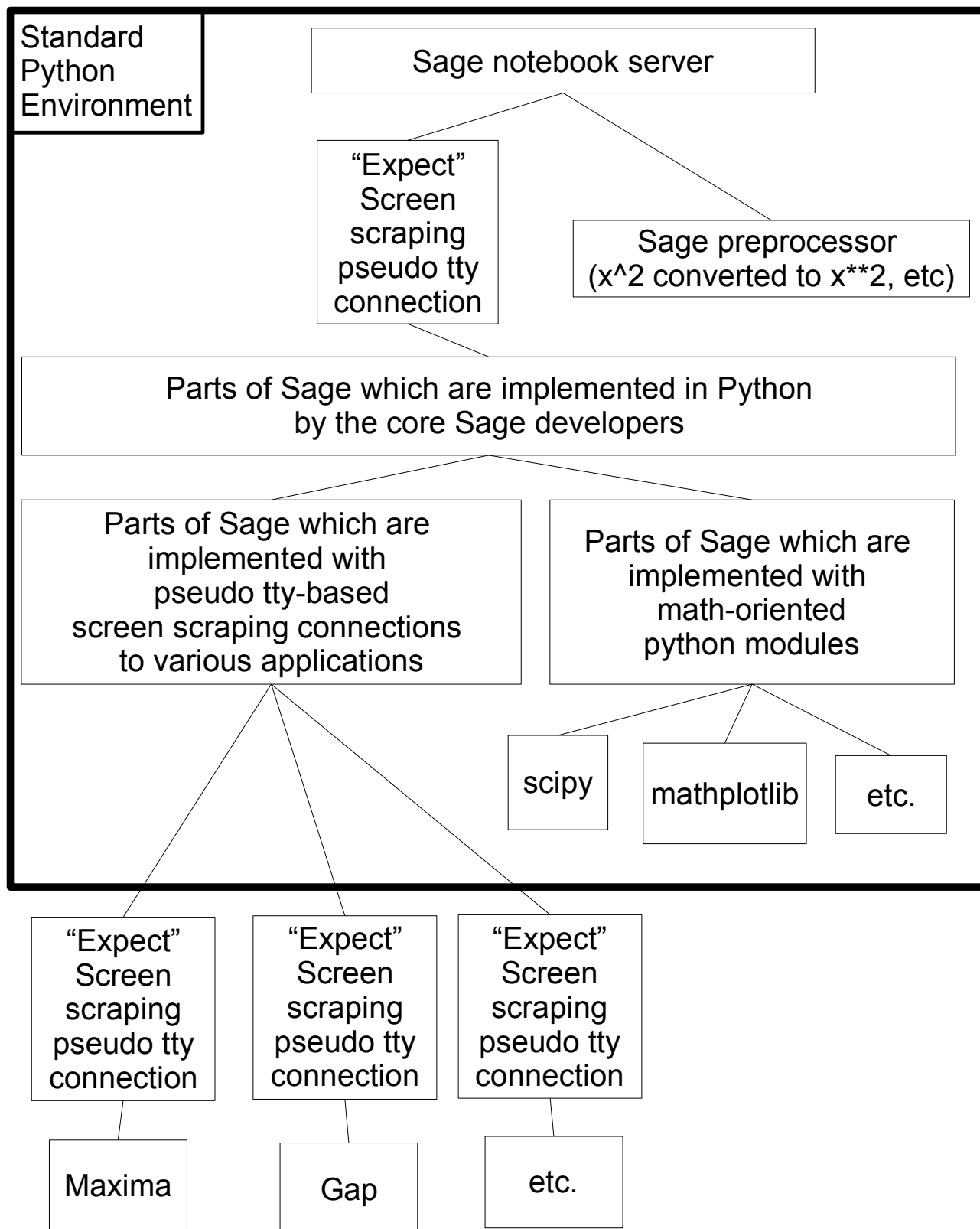


図 13.5: SAGE の構成

3341 **13.3 Linux に基づく SAGE 版**

3342 (まだ...)

3343 **13.4 SAGE の VMware 仮想計算機版(大半の Windows 利用者向け)**

3344 (まだ...)