スクイーク Smalltalk 初めの一歩

市川 雄二 著

「従来からの Smalltalk のユーザーが一般的に考えているのと同様に、私は現在主流の言語よりも Smalltalk のほうが生産性が高いと考えます。」
Martin Fowler『UML モデリングのエッセンス第 3 版』

「残念ながら、Smalltalk は本から学ぶことはできない。」
Simon Lewis『サクサク Smalltalk』

スクイークを始めよう

イントロダクション

みなさんはコンピュータでどんなことをしていますか?

メール? ウェブブラウジング? ゲーム? デジカメ編集? ワープロ? ホームページ作り?

この小文を読んでもらっているということは、スクイーク E トイで遊んだこともある人だと思います。

（まだスクイーク E トイを十分楽しんでいない人は、Thoru Yamamoto さんのスクイークサイト Fun Fun Fun Squeak を是非見てください。）

コンピュータは色々なことができる機械で、コンピュータに何かをする方法を教えることをプログラミングと言います。そしてプログラミングの結果できるものがプログラムです。誰かがプログラミングをしてくれて、そのプログラムがコンピュータにインストールされているから、みんなメールやゲームができるんです。スクイーク E トイで遊んだ人は、車の絵を描いて自分の思うように車を動かしたりしたかと思います。コンピュータに車の動かし方を教えたのだからこれもプログラミングです。

これから一緒にスクイークを使ってプログラミングをしましょう。なぜプログラミングしなきゃいけないのかって? それはきっと楽しいからです（笑）。

これからスクイークの言葉、道具、部品を使いながら覚えていきます。

みなさんに実際に使ってもらうところには、□マークがついていますので、読むだけじゃなく、是非自分でやってみてください。

みなさんおそらくＥトイ用スクイークを使っているかと思います。Ｅトイ用スクイークは、立ち上げた時点ではスクイークＥトイの機能だけ使えるようになっていますが、このスクイークＥトイも、これから紹介する色々な道具もすべて Smalltalk と呼ばれる言葉でプログラミングされています。これから Smalltalk の基本を実際にやってみながら覚えましょう。

□そのために、まず、下のサイトから開発者版スクイークをダウンロードして展開してください。

http://squeakland.jp/plugin/download.html

□さてスクイークを起動してください。

起動するには、SqueakPlugin-dev-527.image を Squeak.exe アイコンの上にドラッグ＆ドロップします。

最初に、日本語版スクイークで、日本語ウェブブラウザを使えるように準備をしましょう。

□ワールド（スクイークでの背景のことです）をクリックしてください。

ワールドメニューが現れます。

□「開く...」をクリックしてください。

メニューが現れます。

□「SqueakMap パッケージ・ローダー」をクリックしてください。

しばらくすると英語で質問されるのですべて"yes"で答えましょう。

"Squeak Map Package Loader"というウィンドウが開いたら成功です。

□Squeak Map Package Loader の左上の欄に、ScamperM と入力して Enter を押してください。

その下の枠の中で ScamperM が選択されたかと思います。

□ScamerM の行を右クリックして"install"をクリックしてください。

いくつか英語で質問されますが、すべて「はい」と答えてください。

これで日本語対応のウェブブラウザがスクイークにインストールされました。早速使ってみましょう。

□再び、ワールドをクリックしてワールドメニューを出してください。

□「開く...」をクリックしてください。

□"Multilingual Web Browser"（英語で多言語対応ウェブブラウザと書いてあります）をクリックしてください。

ウェブブラウザらしきウィンドウが開きましたか?

これがスクイークのウェブブラウザ Scamper で日本語も読めるように直した ScamperM です。Scamper は、ねずみなどが「走り回る」という意味の英単語です。M は Multilingual など色々な意味を込めて付けました。ちなみにスクイーク(Squeak)はねずみのチューチューという鳴き声のことです。どちらもかわいいネーミングですね。

このサイトを ScamperM で見てみましょう。

□browser:about と書いてある欄を消して、http://ich.homelinux.net:8080/squeak/15 と入力してください。

見えましたか?

ちょっとレイアウトが違いますね・・・いつか頑張って直します。

さて、今までした準備を毎回しなくてもいいように、今の状態を保存しましょう。

□ワールドメニューから「別名で保存...」をクリックしてください。

「新しいファイル名は？」と聞かれます。

□FirstStep.image と入力して「了解」をクリックしてください。

これで今の状態が保存できました。

次から続きを読むときには、SqueakPlugin-dev-527.image を使わずに、FirstStep.image を

使ってスクイークを起動してください。

ところで、この小文は、あるときはスクイークＥトイに慣れた中学生、あるときは三角関数を理解している高校生、あるときはコンピュータサイエンス専攻の大学生というようにみなさんがこんな人かなという仮定をころころと変えています。これはわたしの未熟さのせいで、関係することすべてをうまく説明することができなかったからです。この小文の中から自分に合うページを見つけて楽しんでいただけたらこれ以上のよろこびはありません。

プレファレンスの変更

普段ワープロやメモ帳をよく使っている人は、テキストのコピー＆ペーストをよく使っているかと思います。スクイークでもコピー＆ペーストを使うことができます。ただ Windows の場合、スクイークの設定を変えないと、普段使っているショートカットキー割り当てとスクイークのショートカットキー割り当てが異なるので、ここでスクイークの設定を変えて合わせておきましょう。

（Windows を使っている人だけ）

□ワールドをクリックしてください。

□（Windows を使っている人だけ）「ヘルプ ... 」をクリックしてください。

「ヘルプ ... 」メニューが現れます。

□"Preferences..."をクリックしてください。

Preferences ウィンドウがが現れます。

□ウィンドウがマウスカーソルにくっついている状態ですので、全体が見える位置でクリックしてください。

□ウィンドウの上のほうに英単語が並んでいますが、"general"をクリックしてください。

□swapControlAndAltKeys の前の□をクリックしてチェックを入れてください。

Windows を使っている人は、これでショートカットキー割り当てを変更することができました。Ctrl+x でカット、Ctrl+c でコピー、Ctrl+v でペーストができます。

[補足]

プログラミングに慣れてくると、ifTrue:, ifFalse:という文字列を入力するショートカットキー（それぞれ、Ctrl+t, Ctrl+f）が便利になるのですが、swapControlAndAltKeys を設定すると、Ctrl+t, Alt+f という具合に非対称な組み合わせになってしまいます。また、Ctrl+s はサーチのつもりが、swapControlAndAltKeys を設定すると、accept という Squeak にとって大変重要なコマンドが実行されてしまいます。

これらを解消したい人は、InputSensor class-defaultCrossPlatformKeys.st をダウンロードして、fileIn してみてください。

スクイークの基礎

計算しよう

コンピュータは計算が得意なので実際に計算させるのも簡単です。早速スクイークに計算させましょう。計算なんて面白くない？まあ少しお付き合いください。

□下の式の末尾をクリックして文字カーソルを移動させてください。

1 + 1.

□右クリック（Mac:Option クリック）するとメニューが現れるので、「評価して表示 (p)」をクリックしてください。

（スクイークは、Windows でも Macintosh でも動きます。ところが、Windows と Macintosh ではマウスボタンの数や特殊キーの名前が違っているので、操作も若干違ってきます。私が Windows を使っている都合で、Windows の操作方法を最初に書いて、違いがある場合には上のように補足していきます。）

式の後ろに結果が表示されましたか？2 と表示されたら成功です。スクイーク E トイで絵を描くより簡単ですね。

今、みなさんは 1 + 1.というプログラムを走らせたことになります。

「プログラムを走らせる」って変な日本語ですね。コンピュータの世界では色々変な言葉が使われますが、まあ少しずつ慣れてください。例えば、1 + 1.の場合、あまりプログラムらしく見えないので、式と呼ぶことがあります。その場合には「プログラムを走らせる」という代わりに「式を評価する」と言います。変な言葉ですがどちらも同じ意味です。

さて、初めてのプログラムを走らせてみて、スクイークは足し算の仕方をあらかじめ知っているということ、しかもその書き方（言葉）はみなさんが算数で習った書き方と同じ書き方だということがわかりました。最後にピリオドがありますね。これはスクイークにとって文の終わりという意味です。英語みたいですね。

では、もう少し複雑な足し算を実行してみましょう。

□下の式の末尾に文字カーソルを移動させて、「評価して表示」してください。

1+2+3+4+5+6+7+8+9+10.

55 と表示されたら成功です。

1 + 1.の時と違って、+の前後にスペースがなくても計算してくれましたね。

□自分で好きな足し算を下に書いて、「評価して表示」してください。

引き算は-、掛け算は*、割り算は/です。早速混ぜて使ってみましょう。

□下の式を「評価して表示」してください。

1 + 2 * 3.

7 のつもりが 9 になってしまいました。算数では掛け算を足し算より先に計算するという規則がありましたが、スクイークは先頭から順に計算します。（このほうが素直ですよね。人間はややこしいルールが好きなようです。）

算数で習ったのと同じ結果を得るためにはかっこを使いましょう。スクイークはかっこの中を先に計算してくれます。

□下の式を「評価して表示」してください。

1 + (2 * 3).

うまく行きましたか？

スクイークは足し算や掛け算のほかに色々な計算できます。例えば、高校で習う三角関数も。

（まだ習っていない人は飛ばしてください。）

□下の式を「評価して表示」してください。

3.1415926 / 2 sin.

sin(π/2) = 1 に近い値を期待したのに、1 より大きな変な数字が出ませんでしたか?

どうやら、スクイークは 3.1415926 / sin(2)を計算を計算してくれたようです。

先ほどの、スクイークは先頭から順番に計算してくれるという説明はこの場合うそでした。スクイークがどんな順番で計算してくれるかはあとで説明しますね。

□下の式を「評価して表示」してください。

(3.1415926 / 2) sin.

今度は 1 が出ましたか?

ワークスペース

ウェブブラウザの中でスクイークのプログラムを動かしてみました。でもほんとに自分のスクイークでプログラムを動かしたのか半信半疑の人もいるかと思います。そんな人のためにワークスペースを使ってみましょう。

□ワールドをクリックして「開く...」をクリックしてください。

□"workspace (k)"をクリックしてください。

ウィンドウが現れましたか?

このウィンドウはワークスペースと呼ばれるウィンドウです。

□ワークスペースの中で好きにタイプしてみてください。

ワークスペースは、このようにテキストエディタになりますし、フォントを変えたり、スタイルを変えたり、色を変えたりすることもできます。ウエブへのリンクを張ることもできます。

ワークスペースはワープロのように使えるのですが、それだけでなく、ワークスペースでは、今までウエブブラウザの中でやったようにスクイークの言葉を実行することができます。（というより、スクイークではどこででもスクイークの言葉を使うことができるのです。）ワークスペースはプログラミングをするときに色々なことを試してみる場所なので「ワークスペース（仕事場）」と呼ぶのでしょう。そう言えばスクイークＥトイには「仕事場」の代わりに「遊び場」という部品がありますね。

ワークスペースに限らず、スクイークのウィンドウは右上の○をクリックすると、最小化します。もう一度押せば元に戻ります。左上の×をクリックすれば消すことができます。

□今開いたワークスペースを最小化して、それから元の大きさに戻してください。

□今開いたワークスペースを消してください。

□ワークスペースを新たに開いてください。

では、このワークスペースを使って、先ほど紹介したようにスタイルを変えてみましょう。

□このワークペースで好きにタイプして、タイプした文字をドラッグしてしてください。

□Alt(Mac:Cmd)キーを押しながらテンキーではない７（アルファベットキーの上の方にある７）を押してください。

選択した文字列が太文字変わりましたか？

□元に戻すには Alt(Mac:Cmd)キーを押しながらテンキーではない０を押してください。

他にも色々文字をいじることができるので、好きに文章を打ち込んで、

http://www.h3.dion.ne.jp/~y.ich/Squeak/squeak-qref.html#TextStyle

をクリックして「テキストスタイルや強勢を変えるキー」という節を見ながら色々なスタイル変更を試してみてください。

計算の仕組み

ところで、みなさんは科学が好きですか？

（今の世の中、あまり好きじゃない人が多いかもしれません…）

科学には色々な分野があります。物理学、化学、生物学。社会科学というものもあります。それぞれの分野で多くの偉い人たちが色々な規則（法則）を見つけているのですが、化学の規則は物理学の規則で説明できるはず、細胞の規則は生化学と物理学の規則で説明できるはず、という具合に、どうしてそんなことが起こるのかということはそれよりも小さなモノの規則で説明できるとふつう考えられています。この時、一回り小さなモノで説明しようと考えることが大切です。細胞の仕組みは電子や原子の規則から決まっていると考えられますが、実際に考えるときに電子や原子１つ１つから考えると、数が多すぎて複雑になってよくわからなくなるからです。

プログラムも同じことが言えます。すべての計算は、PC の Pentium4 とか Mac の PowerPC G5 とか CPU と呼ばれる部品がしているのですが、複雑なプログラムを CPU に命令するところから考えるとよくわからなくなってしまいます。そこでスクイークでは、色々なモノが互いに助け合って計算すると考えます。1 + 1.の場合、実際には CPU が 1 + 1 を計算したのですが、文の先頭の 1 が+ 1 を計算して結果 2 を返すと考えるのです。スクイークの世界では、数字は足し算や掛け算の計算の仕方を知っているモノです。数字をモノと呼ぶのは変な感じがするかもしれません。「モノ」という考え方はスクイークで大切なところなので、これから繰り返して使ううちに慣れてくるかと思います。スクイーク E トイで遊んだ人は、自分の描いた絵が回ったり動いたりできるモノだと実感しているでしょう。

a + b.は、モノ b と一緒に「足して」というお願いをモノ a に送るという意味です。

1+2+3+4+5+6+7+8+9+10.の場合、

先頭の 1 に、2 と一緒に「足して」というお願いを送ると、1 が足した結果 3 を返してくれます。

今度は返ってきた 3 に、3 と一緒に「足して」というお願いを送ることになるので、6 が返る。

返ってきた 6 に、4 と一緒に「足して」というお願いを送ることになるので、10 が返る。

・・・

という計算をします。

さて、1,2,3 といったモノはどんなお願いを受けることができるのでしょうか？それはスクイークの中に設計図があって、そこに書かれています。

みなさんも設計図が書けるようになるといいですね。

ちょっとだけ設計図を見てみましょう。

□ワールドをクリックしてください。

□「開く...」をクリックしてください。

□"browser (b)"をクリックしてください。

難しそうなウィンドウが現れましたか？これはスクイークの設計図を見るためのブラウザという道具です。

□左上のペインで右クリック(Mac:Option クリック）してください。

メニューが現れます。

□"find class... (f)"をクリックしてください。

"Class name or fragment?"という文字入力ウィンドウが現れます。

□Number（数という意味です）と半角アルファベットで入力して「了解」をクリックしてください。

一覧が現れます。

□「数値」をクリックしてください。

隣のペインに Number が赤く表示されましたか？

右のペインには Number が扱うことができるお願いのリストが表示されています。

□リストの中のお願いをクリックしてください。

下のペインにお願いの中身が表示されます。

今は何もわからなくてもいいので、いくつかお願いの中身をのぞいて下さい。

これらがスクイークでの設計図です。

設計図の説明はまたあとでしますね。

[補足]

最近のコンピュータは計算はCPUだけではなく、グラフィックボードと呼ばれる画面表示用のボードでもしています。

変数

さて数字を使って計算することができるようになりました。

変数を使うと段々プログラムらしくなってきます。

□下の2行をドラッグしてくだしさい。

```
a _ 2.
a _ a * 3.
```

2行が緑色になりましたか?

□「評価して表示」してください。

2つ以上の行をドラッグして選択してから「評価」すると、スクイークは複数行を続けて一度に実行してくれます。

_は左辺の変数が右辺のモノを指し示すようにするという意味です。見慣れない文字ですがアンダーバーをタイプすれば現れます。1行目を実行すると、変数aはモノ2のように振舞うようになります。

2行目を実行すると、

まず、右辺を実行します。変数aが指し示すモノ（今の場合2）に、モノ3と一緒にお願い"*"を送ります。モノ2は計算の結果6を返します。

それから、変数aがモノ6を指し示すようにします。

□「評価して表示」した結果6を消して、もう一度上の2行目を実行してください。

□「評価して表示」した結果を消して、さらにもう一度上の 2 行目を実行してください。

6, 18, 54, ...とどんどん大きくなっていくのが確認できましたか？。スクイークが変数の内容を覚えているのでだんだん大きくなります。

今まで変数、変数と言ってきましたが変数は数以外のものを示すことができます。例えば、

□下の 2 行を選択して「式を評価 (d)」してください。

```
a _ GraphMorph new.
a openInWorld.
```

三角関数のグラフが表示されましたか?

「評価して表示 (p)」と「式を評価 (d)」の違いは、最後の結果を表示はするかしないかだけです。

設計図 GraphMorph に new というお願いを送ると、新しいグラフを作ってくれます（new というのは新しいという意味です）。変数 a は（先ほどの数字じゃなくて）そのグラフを指し示すことになります。

変数 a に openInWorld というお願いを送ると、変数 a の指し示すグラフが openInWorld というお願いを受けてワールドの中にグラフを表示します。変数 a に色々なお願いを送ることでこのグラフを操作することができます。

この場合 a は数じゃないので変数と呼ぶのはちょっと変ですね。スクイークの変数は数学の変数と違うので、参照とか呼ぶほうがたぶんよいのですが、みんな変数と呼ぶので気にせず合わせておきましょう。

作ったグラフはスクイーク E トイの絵と同じように扱うことができて、ホイールをクリック(Mac:Cmd クリック）するとハロ（ウィンドウの周りの丸いアイコンたち）が現れます。×ハロをクリックすれば消すことができます。

変数名

ずっと変数にaを使いましたが、変数の名前はbでもcでも漢字やひらかなでもいいです。でも1は変数名に使えません。だって1は数字の1というモノを表しているから。こういうのをなんだか難しくリテラルと言います。また、いくつかの単語はスクイークが使っているので変数名には使えません。こういうのを予約語と言います。スクイークでは、以下の単語が予約語になっています。

false, nil, self, super, thisContext, true

それから、設計図の名前も変数名には使えません。

使えない名前はスクイークが「使えないよ」と教えてくれるので、心配せずに覚えやすい色々な名前を使いましょう。2つほど覚えてください。

最初は小文字で始める。

これはスクイークのルールです。大文字で始まる変数名と小文字で始まる変数名は役割が違うのでみなさんは小文字で始まる変数名を使ってください。

複数の単語を繋げる時には、単語の頭を大文字にする。

これはスクイークプログラマがみんな従っている慣例です。上のthisContextとかその例です。

リテラル

リテラルというのは、「文字通り」という意味です。1は文字通り数字の1だから数字リテラルです。

数字リテラルの例: 1234, 12345678901234567890, 3.14

文字のリテラルは、対応する文字の前に$をつけることで書くことができます。

文字リテラルの例: $x

数字と違って、xと書くと変数になってしまうので頭に$をつけてリテラル（xという文字そのもの）であることをスクイークに教えます。

文字列のリテラルは、対応する文字列を'で囲むことで書くことができます。

文字列リテラルの例: 'abc'

リテラルの書き方についてもっと色々知りたい人は、

http://www.h3.dion.ne.jp/~y.ich/Squeak/squeak-qref.html#SyntaxLiterals

を見てください。（このページの下のほうに「リテラル（定数表現）という節があります。）

1 は数字リテラルなのですが、計算の仕組みで 1 はモノだという説明をしました。スクイークはリテラルのモノをあらかじめ準備しています。だから、

a _ 1.

というように、あらかじめ持っているモノ 1 を変数 a が指し示すように、という文が書けるのです。

a _ a * 2.

という文を何度も実行すると、変数 a は、スクイークがあらかじめ持っているモノ 2, 4, 6, ...を順に示すように変わっていくことになります。

グラフを作った時は

a _ GraphMorph new.

と書きました。new というメッセージから想像できるように、スクイークは設計図 GraphMorph に従って作られたモノはあらかじめ持っていないので、new というお願いを実行した時に作って、作ったモノを変数 a が指し示すことになります。

まとめると、リテラルはスクイークがあらかじめ持っているモノを文の中に書く時の表現と言うことができます。

条件文

プログラミングには変数の他に条件文が大切です。みなさんの使っているソフトウエアでも、「ユーザがクリックしたら」「画面のはしまで行ったら」「ある時刻になったら」などなどたくさんの条件文を実行しています。

簡単な例から見てみましょう。

□下の 3 行を選択して、「評価して表示」してください。

```
| b |
b _ 3.
b <= 10 ifTrue: ['10 以下'.] ifFalse: ['10 より大きい'.].
```

2 行目は、b が 10 以下かそうでないかを調べて、結果ををを文字列で返す文です。

□変数 b を色々変えて試してみてください。

この文を 1 つ 1 つ見ていくと、

まず、<= 10 は、10 以下かそうでないか聞くお願いです。数字はこのお願いに対して、予約語で見た true か false かどちらかを返します。true は正解という意味のモノです。false は間違いという意味のモノです。

次の ifTrue: ['10 以下'.] ifFalse: ['10 より大きい'.]も 1 つのお願いです。 2 つのお願いに見えますが、これ 2 つで 1 つのお願いです。

スクイークでは、 1 つ一緒に渡したいモノがあるお願いは、

xxx: *渡したいモノ*

という感じに、お願いの名前に:をつけて書きます。

1 つのお願いで 2 つ以上渡したいモノがあるお願いは、

xxx: *渡したいモノ 1* yyy: *渡したいモノ 2* zzz: *渡したいモノ 2 ...*

という書き方をします。

ifTrue:と ifFalse:はお願いする時に['10 以下'.]というモノと['10 より大きい'.]というモノ 2 つを渡したかったのでした。

true や false は、このお願いを受け取ると、true なら ifTrue:のあとの「モノを実行」し、false なら ifFalse:のあとの「モノを実行」します。今まで、分や式を実行してきたので、「モノを実行」って意味がよくわかりませんね。あとで説明しますので、今は、

正しいか間違っているか調べる式

ifTrue: [正しい時実行する文]

ifFalse: [間違っている時実行する文].

という形で条件文が書けると覚えてください。「モノを実行」と一緒に説明することになりますが、かぎかっこ[ ]を忘れないことがとても大切です。

b <= 10 ifTrue: ['10 以下'.] ifFalse: ['10 より大きい'.].

は、

b <= 10

ifTrue:

['10 以下'.]

ifFalse:

['10 より大きい'.].

というように行をかえて書いても 5 行すべてを選択すればスクイークは気にせず実行してくれます。

true や false は、ifTrue:ifFalse:以外にも ifTrue:、ifFalse:、ifFalse:ifTrue:といったお願いを受け取ることができます。ifTrue:ifFalse:と ifFalse:ifTrue:は渡すモノの順序が変わるだけで同じことをしてくれます。ifTrue:と ifFalse:はどんなことをしてくれるか想像つきますよね。お願いの名前からどんなことをしてくれるか想像することはスクイークのプログラミングではとても大切なことです。

条件文を書くには、ifTrue:ifFalse:以外に、正しいか間違っているか調べるお願いを知る必要がありますが、数字に関係する場合、みなさんが算数や数学で習った多くの記号が使えます。

例えば、

= 左辺が右辺と等しいか調べる

< 左辺が右辺より小さいか調べる

<= 左辺が右辺以下か調べる

> 左辺が右辺より大きいか調べる

>= 左辺が右辺以上か調べる

などがあります。

□下に好きな条件文を書いて、実行してみてください。

くり返し文

条件文の兄弟姉妹にくり返し文があります。

前に、

1 + 2 + 3 + 4 + 5 + 6 + 7 + 8 + 9 + 10.
を実行しましたが、同じ計算をくり返し文を使ってやってみましょう。

□下の 4 行をドラッグして、「評価して表示」してください。

```
| sum count |
sum _ 0. count _ 1.
[count <= 10] whileTrue: [sum _ sum + count. count _ count + 1.].
```

sum.

だいぶプログラムっぽくなってきました。

最初に、合計を覚えておく変数 sum と何回くり返したか覚えておく変数 count を用意しました。

改行がなくても.（ピリオド）があればスクイークはこの行には文が 2 つあるとわかってくれます。

（正確にいうとスクイークは改行をスペースと同じようにみなします。）

2 行目は、count が 10 になるまで繰り返す式です。

[続けるための条件式]

whileTrue: [続ける時実行する文].

と覚えてください。前の条件文と違って、先頭の式にもかぎかっこがつくことに注意してください。

「続ける時実行する文」は、今の場合 2 つあって、sum に count を加える文と count に 1 を加える文です。

繰り返すたびに count が 1 ずつ増えていくことが想像できますか？ sum は 1 ずつ増えていく数を足し合わせることになるので、ちょうど 1 + 2 + 3 +, ...を計算することになります。

インスペクタとデバッガ

この足していく様子をスクイークの道具インスペクタとデバッガを使って見てみましょう。

□新たにワークスペースを開いてください。

□下の 3 行をコピーしてワークスペースにコピーしてください。

```
sum _ 0. count _ 1.
[count <= 10] whileTrue: [sum _ sum + count. count _ count + 1.].
sum.
```

□コピーした 3 行を選択して「評価して表示」してください。

さて、

□このワークスペースの上でAlt（Mac:Cmd）キーを押しながらクリックしてください。

（ホイールマウスを使っている人はホイールクリックでもできます。）

ハロが現れます。

□左上2つめのメニューハロをクリックしてください。

メニューが現れます。

□「デバッグ...」までマウスカーソルを移動して、ポップアップしてくるウィンドウの"inspect model"をクリックしてください。

"Workspace"というタイトルのウィンドウが現れましたか？

これがインスペクタです。インスペクタは「調べるモノ」という意味です。今の場合、このワークスペースの情報をリアルタイムで表示してくれます。

□左のペインの中の"bindings"をクリックしてください。

"bindings"が赤くなって、右のペインにcountとsumの文字が見えるかと思います。

これで変数countとsumが今いくつなのか見ることができるようになりました。

次にデバッガを起動します。

ちなみにデバッガは、英語のdebuggerをカタカナ読みしたものです。debuggerはdebugするモノという意味です。debugはbugを取り除くという意味です。bugは虫という意味ですが、コンピュータ業界では不具合のことを意味します。ということはデバッガは不具合を取り除くモノという意味になりますが、実際には不具合があっても残念ながら取り除いてくれません。デバッガがやってくれることは、スクイークがふつう一気に実行するところを、今どこを実行しているか示しながら一文一文実行してくれます。不具合がなくても使えるし不具合の原因を見つけるのにとても便利な道具です。

さて、デバッガを起動しましょう。

□コピーした3行を選択して、右クリックし、「評価してデバッグ」をクリックしてください。

赤っぽいウインドウが現れましたか。これがデバッガです。

真ん中下のペインに上の 3 行が見えるかと思います。（改行とか少し違うところがありますね。）

緑(_ 0)の部分がありますが、これはスクイークが次に実行する部分を示しています。

□先ほど出したインスペクタの sum に注意しながら、デバッガの真ん中上の"Step"ボタンを押してください。

インスペクタの sum の値が 0 になって、デバッガの緑が次の場所に移ったことが確認できましたか?

"Step"ボタンはこのように、割り当てやお願いを 1 つ 1 つ実行してくれます。

さて、くり返し"Step"ボタンを押して、インスペクタやデバッガを観察したくなったかと思いますが、その前に 1 つ呪文を覚えてください。

スクイークは、時々プログラムを実行するのに夢中になるときがあります。そんな時はどこをクリックしても反応してくれなくなるのですが、そんな時には、Alt(Mac: Cmd)を押しながら.（ピリオド）をタイプしてください。そうするとたいていの場合、スクイークはみなさんに気がついて、プログラムの今実行しているところをデバッガで表示して止まってくれます。

さて、

□くり返し"Step"ボタンを押して、インスペクタやデバッガを観察してください。スクイークが応答しなくなるまで"Step"ボタンを押してください。スクイークが応答しなくなったら、Alt(Mac:Cmd)を押しながら.（ピリオド）を押してください。

デバッガやインスペクタは×ボタンを押して消すことができます。

インスペクタとデバッガはとても便利な道具なのでここでくり返し使って慣れてください。

例えば、Step"のほかに"Proceed", "Restart", "Send", "Through", "Full Stack", "Where"というボタンが見えますが、それぞれをクリックするとどんなことが起こるか試してください。

さらにくり返し文

1 から 10 までの和は、他にもいくつかの書き方が考えられます。

例えば、

| sum |

```
sum _ 0. count _ 1.
10 timesRepeat: [sum _ sum + count. count _ count + 1.].
sum.
```

これは、

```
繰り返す回数
    timesRepeat:
        [繰り返す文].
```

と覚えてください。

こんな風にも書けます。

```
| sum |
sum _ 0.
1 to: 10 do: [:each | sum _ sum + each. ].
sum.
```

これは、

```
ある数字 to: 別の数字
    do:
        [:ある数字から別の数字が順に入る変数名 |
        繰り返す文].
```

と覚えてください。変数には好きな名前をつけてもいいですが、繰り返す式の中で同じ変数名を使います。ちなみに each というのは「それぞれ」という意味の英単語です。1 から 10 までのそれぞれの数字が入るので変数名を each としました。

inject: into:というお願いを使うと、sum に最初の値を割り当てる文まで含んだ一文を書くことができます。

```
(1 to: 10) inject: 0 into: [:sum :each | sum _ sum + each].
```

これは、

```
(ある数字 to: 別の数字)
    inject: 次のブロックの一つ目の変数に最初に割り当てるモノ
    into:
```

[:一つ目の変数名 :ある数字から別の数字が順に入る変数名 |

繰り返す文].

と覚えてください。

どうして同じ計算をするのに、こんなにたくさんの書き方があるのでしょうか?

言いたいことをうまく表すために、色々な書き方が用意されていると考えてください。日本語の文章でも、ふさわしい言葉を使った文章は、読むにはある程度の知識が必要になりますが、その代わりに書いた人が伝えたいことを正確に伝えてくれますよね。

でも、今は上の中から気に入った文章を 1 つ覚えたら十分ですよ。

ブロック

条件文やくり返し文の中でかぎかっこ[ ]がたくさん出てきました。かぎかっこで囲まれた式はブロックと呼ばれるちょっと変わったモノです。

文を書いて「式を評価」すると、スクイークはその文を実行してくれます。ところが、文を[ ]で囲むとスクイークは文だと思わずモノだと思って、その場ではそっとしておきます。

□下の 3 行を「評価して表示」してください。

```
a _ 1.
b _ [a _ a + 1.].
a.
```

2 行目に a _ a + 1 という文があるけれど、スクイークは実行しなかったので a は 1 のままです。

□下の 2 行を「評価して表示」してください。

```
b value.
a.
```

2 と表示されました。ブロックは書いたところでは中身の文は実行されないけど、value というお願いを受け取ると中身の文を実行するというモノなのでした。

お願いと一緒にモノを送ってもらうブロックも書くことができます。

[:i | sum _ sum + i] の先頭はこのブロックを実行する際にモノを 1 つ必要とすることを示しています。このブロックを評価するには、value: <モノ>というお願いを送ります。value の後に:（コロン）がついていることに注意してください。

今までブロックを条件文の中で使ってきましたが、ブロックを使うと関数を作ることができます。三角数を返すブロックを作ってみましょう。三角数というのは、数列の一種で、n 番目の三角数は、1 から n までの数を足した値のことです。「計算しよう」で 1+2+3+4+5+6+7+8+9+10 を計算しましたが、これは 10 番目の三角数です。

□下の 6 行を「式を評価」してください。

```
triangleNumber _
	[:n |
		sum _ 0.
		1 to: n do: [:i | sum _ sum + i].
		sum.
	].
```

変数 triangleNumber が三角数を返すブロックを指し示すようにしました。

□下の文を「評価して表示」してください。

```
triangleNumber value: 10.
```

□下の文を「評価して表示」してください。

```
triangleNumber value: 100.
```

このブロックを使わずに例えば 500 番目の三角数を計算させようとすると、

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+28+29+30+31+32+33+34+35+36+37+38+39+40+41+42+43+44+45+46+47+48+49+50+51+52+53+54+55+56+57+58+59+60+61+62+63+64+65+66+67+68+69+70+71+72+73+74+75+76+77+78+79+80+81+82+83+84+85+86+87+88+89+90+91+92+93+94+95+96+97+98+99+100+101+102+103+104+105+106+107+108+109+110+111+112+113+114+115+116+117+118+119+120+121+122+123+124+125+126+127+128+129+130+131+132+133+134+135+136+137+138+139+140+141+142

+143+144+145+146+147+148+149+150+151+152+153+154+155+156+157+158+159+160+161+162+163+164+165+166+167+168+169+170+171+172+173+174+175+176+177+178+179+180+181+182+183+184+185+186+187+188+189+190+191+192+193+194+195+196+197+198+199+200+201+202+203+204+205+206+207+208+209+210+211+212+213+214+215+216+217+218+219+220+221+222+223+224+225+226+227+228+229+230+231+232+233+234+235+236+237+238+239+240+241+242+243+244+245+246+247+248+249+250+251+252+253+254+255+256+257+258+259+260+261+262+263+264+265+266+267+268+269+270+271+272+273+274+275+276+277+278+279+280+281+282+283+284+285+286+287+288+289+290+291+292+293+294+295+296+297+298+299+300+301+302+303+304+305+306+307+308+309+310+311+312+313+314+315+316+317+318+319+320+321+322+323+324+325+326+327+328+329+330+331+332+333+334+335+336+337+338+339+340+341+342+343+344+345+346+347+348+349+350+351+352+353+354+355+356+357+358+359+360+361+362+363+364+365+366+367+368+369+370+371+372+373+374+375+376+377+378+379+380+381+382+383+384+385+386+387+388+389+390+391+392+393+394+395+396+397+398+399+400+401+402+403+404+405+406+407+408+409+410+411+412+413+414+415+416+417+418+419+420+421+422+423+424+425+426+427+428+429+430+431+432+433+434+435+436+437+438+439+440+441+442+443+444+445+446+447+448+449+450+451+452+453+454+455+456+457+458+459+460+461+462+463+464+465+466+467+468+469+470+471+472+473+474+475+476+477+478+479+480+481+482+483+484+485+486+487+488+489+490+491+492+493+494+495+496+497+498+499+500.

と書いて、「評価して表示」しないといけないのですが、今、みなさんは triangleNumber というブロックを作りましたので、

triangleNumber value: 500.

とだけ書いたら済むのです。これがプログラミングの威力です！

素数

1, 2, 3, 4, ...という数は自然数と呼ばれています。自然数の中には、その数より小さい自然数の掛け算で書けるものがあります。例えば、4 は 2*2 と書けます。逆にこのように自分より小さな自然数の掛け算で書けない数字もあります。例えば、2, 3, 5, 7, 11 です。このように自分より小さな自然数の掛け算で書けない数字を素数と言います。

素数でない自然数は、自分より小さな自然数の掛け算で書けるのですが、実は、自分より小さな**素数**の掛け算で書けることがわかっています。

4=2*2, 6=2*3, 9=3*3, 10=2*5, 12=2*2*3, ...

素数はとても不思議な数字たちです。その魅力にとり憑かれて、昔から今に至るまで大勢の人が研究を続けています。

さて、ここでは、素数を集め続けるプログラムを書いてみましょう。

□ワールドメニューの「開く・・・」からトランスクリプトを開いてください。

□以下の 5 行を選択して「式を評価」してください。

```
prime _ OrderedCollection new.
index _ 2.
[prime detect: [each: (index \\ each) = 0] ifNone: [prime add: index. Transcript show: index; cr].
index _ index + 1.] repeat.
```

トランスクリプトに素数がすごい速さで表示されるようになります。

プログラムを止めるには、

□Alt キーを押しながら.（ピリオド）を押してください。

専門用語

どうですか？スクイーク楽しいですか？楽しいと思ってもらえるといいのですが・・・

数字ばかりで面白くないかもしれませんね。基本が一通り説明できたらスクイークＥトイと合わせてプログラムを作ろうと思いますので、もう少し我慢してくださいね。

今まで、スクイークの言葉（お願いの仕方とか_やら.やら;やら）は色々覚えて、道具（ワークスペース、ブラウザ、インスペクタ、デバッガ）もいくつか使いました。部品（数字、グラフ）も少し使いました。

ここまで、できるだけ普段からよく使われる言葉で説明してきたつもりなのですが、どんな分野も専門用語というのがあります。野球の「ストライク」「バット」とかって普段は使いませんよね。ああいうのが専門用語です。みなさんがスクイークについてほかの人とスムーズに話ができるように、スクイークの専門用語を書いておきます。難しいと思うかもしれませんが、既にみなさんが知っているものを別の言葉で言い換えるだけです。

今まで使ってきた言葉と専門用語

左側に今まで使ってきた言葉、右側にスクイークの専門用語を書きます。

文	=	式
実行	=	評価
モノ	=	オブジェクト
お願い	=	メッセージ
お願いと一緒に渡すモノ	=	引数（ひきすう）
指し示す	=	代入する

設計図　＝　クラス

設計図通り動くモノ　＝　インスタンス

メッセージとお願いが一緒と書きましたが、実はちょっとややこしいところです。

スクイークでは、正確には、

受け取るモノ＋お願い＋お願いと一緒に渡すモノ　＝　メッセージ

となります。

受け取るモノ　＝　レシーバ

お願い　＝　メッセージセレクタ

という専門用語が用意されています。

しかし、専門家も（短く言うために）メッセージセレクタをメッセージと言いますので、みなさんもあまり気にしないでください。

規則のまとめ

専門用語を覚えたら、

http://www.h3.dion.ne.jp/~y.ich/Squeak/squeak-qref.html#Syntax

を見てみてください。

スクイークの規則が書いてあります。言葉が難しく思えるかもしれません。全部わかる必要はないので、覚えた規則を探してみてください。

メッセージの評価する順番について説明していなかったので、ここで説明します。

メッセージには、３つの種類があります。

引数がないメッセージ

例: GraphMorph new.の new

引数が１つで記号で表すメッセージ

例: 1 + 1.の + 1

引数が１つ以上で言葉で表すメッセージ

例: a log: 10 の log: 10、a = b ifTrue: [] ifFalse: []の ifTrue: [] ifFalse: []

このメッセージの特徴はメッセージセレクタの最後に:（コロン）が付くことです。コロンがついていたら引数があると覚えましょう。

複数のメッセージが連なる式を評価する時、スクイークは、

最初に()でくくった式を評価します。

次に引数がないメッセージを左から順に評価します。

次に引数が 1 つ以上で記号で表すものを左から順に評価します。

最後に引数が 1 つ以上で言葉で表すものを左から順に評価します。

「計算しよう」で試した 3.141529 / 2 sin.は、sin が引数がないメッセージだから先に、スクイークは先に 2 sin を計算したのでした。

条件文の仕組み

```
| b |
b _ 3.
b = 10 ifTrue: ['10 以下'.] ifFalse: ['10 より大きい'.].
```

「条件文」の節で上の式を覚えましたが、スクイークがこの文章をどんな風に実行するのか見てみます。今覚えた専門用語を使ってみますね。

変数 b が引数 10 と一緒にメッセージ=を受け取ると、変数 b に割り当てられている 3 が 10 以下かどうか調べて、正しいという意味でオブジェクト true を返します。

次に返されたオブジェクト true は 2 つの引数['10 以下'.]と['10 より大きい'.]と一緒にメッセージ ifTrue:ifFase:を受け取ります。true は ifTrue か ifFalse かと聞かれたらみなさん ifTrue を選びますよね。スクイークも同じように考えて ifTrue:の後のブロック['10 以下'.]を実行します。そう、['10 以下'.]はブロックなので必要な時に実行できるんです。

さて、ブロック['偶数']やブロック['奇数']はどこで評価されるのでしょうか?

□下の行を「評価してインスペクト」してください。

```
true.
```

インスペクタウィンドウが開きます。

□self/all inst vars と書かれた枠の中で右クリック（Mac:Option クリック）して「階層ブラウザを開く」を選択してください。

□階層ブラウザの左端の枠の ifTrue:ifFalse:をクリックしてください。

以前にブラウザは設計図を見るための道具と言いました。今、true の設計図を見ているのです。

下の枠に ifTrue: ifFalse:の中身が表示されます。アルファベットなのでわかりにくいかと思いますが、trueAlternativeBlock にメッセージ value を送っていることが確認できましたか？ true は ifTrue: ifFalse:メッセージを受け取ると、ifTrue:の後のブロックに value を送ることがわかりました。

メッセージを実行する時の具体的な中身をメソッドと言います。メソッドは「方法」という意味です。

エラー通知とワークスペース変数

「ブロック」の節で、三角数を計算するブロック triangleNumber を作りました。

ここで、

□新たにワークスペースを開いて、下の行を「評価して表示」してください。

triangleNumber value: 10.

"MessageNotUnderstood: value:"というウィンドウが飛び出てきて、triangleNumber を計算してくれなくなってしまいました。

ところで、今現れたウィンドウは、これから何度もお世話になるエラー通知です。"Proceed", "Abandon", "Debug"の 3 つのボタンがあります。"Proceed"ボタンをクリックするとエラーを無視して続けて評価してくれます。"Abandon"ボタンをクリックすると評価を中止します。"Debug"ボタンをクリックすると、デバッグウィンドウが現れて、エラーになるまでどんなメッセージの受け渡しがされたのかを見ることができます。

□"Abandon"ボタンをクリックしてください。

"MeesageNotUnderstood: value:"というのは「value:というメッセージは知りません」という意味です。ブロックは value:というメッセージの実行の仕方を知っていますが、どうやらこのワークスペースの triangleNumber value:の実行の仕方を知らないようです。

うすうす気がついていると思いますが、スクイークは英語で作られています。そう、日本人にとって不幸なことに、作った人たちが英語を話す人たちだったのです・・・メニューが日本語だったり、こうやって日本語を読み書きできるのは、スクイークを日本語化してくれた人たちがいるおかげなのです。どちらの人たちにも感謝感謝です。

元に戻って、念のために、

□ワークスペースの triangleNumber を「評価して表示」してください。

nil と表示されましたか？nil はスクイークの予約語の 1 つで、UndefinedObject（「定義されていないオブジェクト」という意味）というクラスのインスタンスです。

スクイークは代入される前の変数には nil を代入するようにしています。

このワークスペースでは変数 triangleNumber は nil ということは、「ブロック」の節で作った triangleNumber はどこへ行ってしまったのでしょう？

変数はワークスペース毎に別々に用意されます。「ブロック」の節での変数 triangleNumber とここで書いた triangleNumber は名前を同じにしても全く別のものなのです。

「ブロック」の節で作った triangleNumber がここでは使えないのですが、*や ifTrue:ifFalse:はここでも使うことができます。それは、+や ifTrue:は設計図にメソッドとして書かれているからなのです。

メソッドを作る

三角数を計算するブロックを変数に代入するやり方はワークスペース毎に毎回用意する必要があることがわかったので、スクイークのどこででも三角数が計算できるように、メソッドを作りましょう。

まずメソッドを追加するクラス（設計図）を決めます。

「ブロック」の節で作った triangleNumber は

triangleNumber value: 数

という使い方をしました。数を渡してもらえば三角数を計算してくれます。もう少し正確に考えると、整数を渡してもらえば計算してくれます。ということは、整数に triangleNumber というメッセージを渡したら計算してくれたら嬉しいですよね。そのためには整数クラス(Integer)にメソッド triangleNumber を追加したらいいのです。

さあやってみましょう。

□ワールドメニューの中の「開く...」メニューをからブラウザを開いてください。

□左はしのペインの中で右クリックして、"find class... (f)"をクリックしてください。

□integer と入力して「了解」ボタンをクリックしてください。

□一覧が出るので一番上の Integer をクリックしてください。

□右から 2 番目の枠で"mathematical functions"をクリックしてください。

（"mathematical functions"というのはメソッドを探しやすいようにつけた分類名です。triangleNumber を mathematical functions（数学的な関数）に分類してみます。）

下のペイン（枠）に以下のようなテキストが表示されます。

```
message selector and argument names
    "comment stating purpose of message"

    | temporary variable names |
    statements
```

これはメソッドの書き方を示すテンプレートです。訳してみると、

```
メッセージセレクタと引数の名前
    "メッセージの目的を書くコメント"

    | 一時変数名 |
    文
```

という感じです。

□ブラウザのテンプレートを全部消して、そこに下の 5 行をコピー＆ペーストしてください。

```
triangleNumber
	| sum |
	sum _ 0.
	1 to: self do: [:i | sum _ sum + i].
	^ sum.
```

□ 5 行をコピー＆ペーストしたペインの中で右クリック（Mac: Cmd クリック)して「了解 (s)」をクリックしてください。

初めてスクイークの設計図に手を入れたときには、"Please type your initials:というウィンドウが現れますので、自分だとわかる何か（イニシャルでもハンドルネームでもなんでもＯＫ）を入力しましょう。

うまく行けばすぐに入力したペインの回りの赤色が消えます。

もしかしたら、消える代わりに "Period or right bracket expected ->"というテキストが挿入されたり、何かポップアップウィンドウが現れたりしたかもしれません。これはスクイークがうまく読めない文があると聞いてきているんです。今の場合、タイプミスかと思いますので、テキストが挿入されたらその部分は削除して、ウィンドウが現れたら「取り消し」を押してもう一度上の式と入力したペインの内容を 1 文字ずつ比較してみてください。

何も現れず枠の赤色が消えたらスクイークはメソッドを取り込んでくれました。（メソッドをコンパイルしたと言います。）

これで、スクイークのどこででも整数はメッセージ triangleNumber を受けて計算できるようになったんですよ。

□下の行を「評価して表示」してください。

```
500 triangleNumber.
```

うまくいきましたか？できちゃいましたね。

□別のワークスペースを開いて、そこで"1000 triangleNumber."を「評価して表示」してみてください。

もうどこでも三角数が計算できるようになりました。

今まではスクイークは、ワークスペースに書かれた文を評価してと言われたときにその文をコンパイルして実行していたのですが、ついにみなさんスクイークの設計図に手を入れることで、クラス Integer のインスタンスに triangleNumber という言葉とその意味を覚えさせたのです！

「ブロック」で書いた triangleNumber とメソッド triangleNumber を比較しておきましょう。

・ブロック

```
triangleNumber _
    [:n |
        sum _ 0.
        1 to: n do: [:i | sum _ sum + i].
        sum.
    ].
```

・メソッド

```
triangleNumber
    | sum | "一時変数宣言"
    sum _ 0.
    1 to: self do: [:i | sum _ sum + i]. "self はメッセージを受け取った自分"
    ^sum. "sum の指し示すオブジェクトを返す"
```

主な違いのある行に" "で囲んで説明文を書きました。スクイークは" "の間の文字を無視するので、先ほどのコードの代わりに、上の 5 行をシステムブラウザに入力しても全く同じ動きをします。

| は一時変数の宣言です。 ワークスペースで変数を使うときは、変数はそのワークスペースの中でだけ有効だと説明しました。メソッドは「システム」ブラウザで作ったことからも想像できるかもしれませんが、スクイーク全体に影響を与えます。そこで変数がメソッドの中だけで使う変数の場合、そのことをスクイークに教える必要があります。これが一時変数の宣言です。

self はメソッドを受けたインスタンス自体を示します。

^は右のオブジェクトをメッセージの返り値として返すという意味です。キーボードの

キャロット（ひらがなの「へ」のキー）をタイプすると入力できます。

今、スクイークは triangleNumber という新しいメソッドを覚えてくれたのですが、これを次にスクイークを起動しても使えるようにするには「保存」をする必要があります。

□ワールドをクリックしてワールドメニューを出してください。

□「保存」をクリックしてください。

次回から FirstStep.image を Squeak.exe にドラッグ＆ドロップすることで triangleNumber を使うことができます。

再帰呼び出し

（この節は難しそうでしたら読み飛ばしてください。）

メソッド triangleNumber を作った時に、mathematical functions というカテゴリに追加しました。

□システムブラウザでもう一度このカテゴリを見てください。

□factorial というメソッドがあるので、それをクリックしてください。

factorial は階乗を計算するメソッドです。階乗というのは、例えば、10 の階乗は、

1*2*3*4*5*6*7*8*9*10

です。階乗はこんな具合に、1 から順に掛け合わせます。掛け算と足し算が違うだけで、三角数とよく似ていますね。

システムブラウザの下の部分のペインを見てください。

掛け算と足し算が違うだけのはずなのに、triangleNumber とずいぶん違います。

びっくりするのは、メソッド factorial の中でメッセージ factorial を送っています。何かぐるぐる目が回ってしまいますね。

メソッド factorial は、

10 の階乗は、9 の階乗に 10 を掛けたもの

という性質を使って計算しています。このように計算するのに自分自身を呼び出す使い方を再帰呼び出しと言います。なんだかかっこいいですね。

□triangleNumber も factorial のまねをして変えてください。

うまくできましたか？スクイークに変更を覚えてもらうには、追加した時と同じ方法でいいです。

（右クリック（Mac: Cmd クリック)して「了解」をクリック。）

triangleNumber の変更の一例をこのワークスペースの終わりに書きました。

今度はブロックで再帰呼び出しをしてみましょう。

```
| triangle |
triangle _
	[:n |
		n < 0 ifTrue: [ 'マイナスは入れないで'. ].
		n = 0 ifTrue: [ 1. ].
		n > 0 ifTrue: [ n + (triangle value: (n - 1)). ].
	].

triangle value: 10.
```

□上の 9 行を「評価して表示」してください。

Error: Attemp to evaluate a block that is already being evaluated.

というエラー通知が現れましたか？

「既に評価されているブロックを評価しようとした」と言っているので、どうやらブロックの再帰呼び出しはできないようです。

だんだんメソッドが便利に見えてきましたね。

再帰呼び出しによる triangleNumber の一例

```
triangleNumber
	"Answer the triangle number of the receiver."

	self = 0 ifTrue: [^ 0].
```

self > 0 ifTrue: [^ self + (self - 1) triangleNumber].

self error: 'Not valid for negative integers'

プログラミングの種類

みなさん、メソッドの作り方まで覚えて、かなりスクイークの言葉を覚えてきました。

ここでスクイークでプログラミングをするのに何通りの方法があるのでしょうか。

リテラルとすでにあるメソッドを使って、式を作る。

1 + 1.などの計算がその例です。

すでにあるクラスからインスタンスを新たに作り、すでにあるメソッドを使って式を作る。

GraphMorph new openInWorld.などがその例です。

ブロックを作る。

三角数を計算するブロックを作りましたよね。

メソッドを作る。

すでにあるクラスにメソッド triangleNumber を追加しました。

これ以外にもみなさんスクイークＥトイでプログラミングしていると思います。スクイークＥトイで、モーフを作って、大きさを変えたり、タイルスクリプトを作るというのは、

すでにあるクラスからインスタンスを新しく作る。

インスタンスの性質を変更する。

これは実はインスタンスが持つ変数に、別のインスタンスを新たに代入することです。

そのインスタンスのためにタイルスクリプトを覚えるクラスを作る。

そのクラスにタイルスクリプトに対応するメソッドを作る。

ということをしています。みなさん知らず知らずのうちにクラスを自分で作っていたのですね。

そう、第 6 の方法

6. 新しくクラスを作る

というのがスクイークのプログラミングの一番の醍醐味です。

クラスと継承

さて、triangleNumber というメソッドを作ったのですが、メソッドを作る時にはまかせるクラスを決める必要があることがわかりました。triangleNumber の場合には、ブロックの時の引数だったクラス Integer にしました。このように任せる適切なクラスがすぐ見つ

かったらそこにメソッドを追加することでプログラミングができます。

さて、任せる適切なクラスが見当らない場合には、新しいクラスを自分で作ることになります。

新しいクラスを作る時には、必ず元になるクラスを選びます。

クラスＡの元になるクラスがＢの時、

クラスＡのスーパークラスはクラスＢである。

クラスＡはクラスＢのサブクラスである。

と言います。

スーパーとかサブとか難しそうですが、

Ａさんの親はＢさんである。

Ａさんは Ｂさんの子供です。

と同じような意味です。

今までクラス（設計図）を調べるのにブラウザを使ってきましたが、ブラウザには色々な種類があります。スーパーとサブの関係を調べるのに便利なブラウザは階層ブラウザです。

□下の行をドラッグして、右クリック(Mac: Option クリック）してください。

Integer

□メニューの「さらに ... 」をクリックしてください。

□"browse it (b)"をクリックしてください。

□左から２番目のペインの"Integer"を右クリック（Mac: Option クリック）をしてください。

□メニューの"browse hierarchy (h)"をクリックしてください。

（hierachy は階層という意味です。）

階層ブラウザを使って、クラス Integer を見ることができるようになりましたか?

クラスペインを見てください。Integer のスーパークラスは Number ですね。整数は数の一種だからです。このようにサブクラスはスーパークラスの一種になるように作っていきます。親の性格が子供に遺伝するのと似ていますね。実際、スーパークラスのインスタンスが受け取れるメッセージは、そのサブクラスのインスタンスも受け取れるので、機能が遺伝します。専門用語では、遺伝という代わりに、難しく継承と言います。（王位継承とか時々ニュースとかでたまに聞きますね。）

ということは、新しいクラスを作る時には、スクイークが既に持っているクラスについて知っておくよさそうです。実際、たくさん知っておけば既にあるプログラムを利用できるので効率よくプログラミングができるようになります。

スクイークは一体いくつのクラスを持っているのでしょうか。

実際に調べてみましょう。

階層ブラウザがクラスの親子関係についてよく知っていそうなので、

□ブラウザを開いて、クラス HierarchyBrowser を検索してください。

```
Browser subclass: #HierarchyBrowser
    instanceVariableNames: 'classList centralClass '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Tools-Browser'
```

というコードが見つかりましたか?

このコードが実は新しいクラスを作ったり変更したりする式なのですが、それは後で説明するとして、classList という単語が目を引きます。

classList の中身がわかればクラスの数も数えられそうです。

□ブラウザの中央の並びにある inst vars ボタンを押してください。inst vars は instance variables の略です。インスタンス変数（後で説明します）のリストが現れるので、classList をクリックしてください。

classList を使っているメソッドの一覧が表示するウィンドウが現れます。

それぞれのメソッドをクリックして順に眺めていくと、initAlphabeticListing が簡単そう

です。

その中から classList の代入文の右辺を下にコピー＆ペーストしてきました。

Smalltalk classNames.

Smalltalk は、スクイーク自身が使っている変数（システム変数）です。この変数に classNames というメッセージを送った結果を classList に代入しています。

□上の 2 式を「評価してインスペクト」してください。

インスペクトウィンドウで classList の中身を見ることができます。左の枠に番号がありますね。スクロールさせて最後の番号を確認してください。それがスクイークが持っているクラスの数です。

2000 個近くあります。スクイークは 2000 個近くの設計図を持っているんですね！

先ほど、「たくさん知っておけば既にあるプログラムを利用できるので効率よくプログラミングができるようになる」と書きましたが、2000 個近くのクラスを覚えるのは大変です。でも一度に覚える必要は全くありません。

クラスやそのメソッドを知るのは、語学で単語を覚えるのと一緒です。単語をたくさん知っていれば、うまく言い表せたり簡単に言い表せたりするのと一緒で、クラスやメソッドを知るにつれてうまくプログラミングができるようになります。

でも少しだけでも覚えたら色々なことができるようになるので、最初は上手下手は気にせず、単語を 1 つ 1 つ覚えながらどんどんしゃべって（プログラムして）みましょう。

サブクラスの数

「クラスと継承」でクラスの数を調べました。クラスには親子関係があることを思い出すと、それぞれのクラスの子供の数がどれくらいか知りたくなりませんか？

□ブラウザでクラス Class を見てください。

Class がどんなことができるかメソッドを眺めていくと、ありましたありました。subclasses というメソッドに「レシーバのサブクラスを含む Set を返す」と書いてあります。Set というのは数学の集合のことです。早速使ってみましょう。

Object subclasses.

□上の式を「評価してインスペクト (i)」してください。

今は数を知りたいので、今度は返り値のクラスである Set のメソッドを調べてみましょう。

□システムブラウザでクラス Set を見てください

Class がどんなことができるかメソッドを眺めていくと、size というメソッドが見つかりました。

□下の式を「評価して表示」してください。

Object subclasses size.

これで、Object のサブクラスの個数がわかりました。

以前に Smalltalk というグローバル変数からクラス名のリストを取り出したことを覚えていますか?

Smalltalk classNames
Smalltalk をちょっと調べておくと便利そうです。

□Smalltalk を選択して、右クリック（Mac:Option クリック）して「さらに ... 」をクリックしてください。"browse it (b)"をクリックしてください。

クラス SystemDictionary のブラウザが開きました。Smalltalk は SystemDictionary のインスタンスだということです。

メソッドを眺めてみると、allClasses というメソッドがありました。allClassesDo:というメソッドもありますね。

□Transcript を開いておいて、下の式を実行してみてください。

```
Smalltalk allClassesDo:
    [:each |
    Transcript show: each; space; show: each subclasses size; cr].
```

すべてのクラスのサブクラスの数が出ましたか？

せっかくなので、サブクラスの多い順番に並べたいですね。

□下の式を実行してください。

```
data _ Dictionary new.
Smalltalk allClassesDo:
    [:each |
    data add: (Association key: each value: each subclasses size)].
```

Dictionaryというのは辞書のことですが、スクイークでは、もっと広い意味で、オブジェクトの対の集合を示します。確かに辞書は単語とその意味の対の集合ですね。今、クラスとそのクラスのサブクラスの数という対になったデータを集めたかったのでDictionaryを使いました。dataにadd:しているAssociationはまさに対を意味します。こうやって、キーと値を対にしてDictionaryのインスタンスに追加すると、キーを使って値を知ることができます。

```
data at: Dictionary
```

を「評価して表示」すると、クラスDictionaryのサブクラスの数が表示されます。

□実際に「評価して表示」してみてください。

では、集めたdataを並び替えてみましょう。並べ替えをしてくれるクラスにSortedCollectionがあります。dataをSortedCollectionに変換したモノを作っちゃいましょう。

□下の式を「評価してインスペクト」してください。

```
data asSortedCollection: [:a :b | a value > b value].
```

インスペクタが出てきましたか？上の式にブロックの引数がついていますが、これは、このブロックがtrueになるように並べかえるようにお願いしています。

インスペクタで中身を見ると、どうやら数字の配列のようです。確かに大きな数字の順

番に並んでいます。並び替えはできたようですが、クラスの情報がなくなってしまいました・・・Dictionary を SortedCollection に変換するとキーの情報が消えてしまうようです。（キーは配列のインデックスと一緒なので並び替えるとインデックスの情報がなくなるのは実は当然なのです。）

では、Association を Dictionary で集めずに、Set で集めてみましょう。

□下の式を「評価してインスペクト」してください。

```
data _ Set new.
Smalltalk allClassesDo:
    [:each |
    data add: (Association key: each value: each subclasses size)].
data asSortedCollection: [:a :b | a value > b value].
```

うまく行きましたか?

上の式は、ただ Dictionary を Set に変えただけの式です。これで、インスペクタで中身を見るとちゃんとクラスの情報も見ることができました。

Object のサブクラスは数百とあります。でも次のクラスからはずいぶん数が減ることを確認してください。２０００近くあるクラスのうちほとんどは、

ほとんどのクラスはサブクラスは持たないか、持っても 7 個ぐらいです。でも、Object のように５００個近くサブクラスを持つクラスもあるのです。

この事実は実はとても大切です。あるクラスの機能を使うクラスを使いたいときには、

そのクラスを継承する。（ホワイトボックス）

そのクラスのインスタンスを持つ。（ブラックボックス）

という 2 種類の方法がありますが、多くの場合、2 番目の方法がよい方法になります。だから Object を継承して使いたいクラスのインスタンスを持つクラスがたくさんあるのです。

モーフィック

みなさん基本は習得できても、三角数やらクラスの数やらプログラミングって退屈と思われてしまっているかもしれません。

ここで色々な絵を出すプログラミングに挑戦しましょう。

と言っても、私がすることはみなさんにいいウェブサイトを紹介することだけです。

その前にウェブサイトの内容と対応をつけやすくするため、スクイークの設定ををを少し変えましょう。

□ワールドをクリックしてワールドメニューを出してください。

□「フラップ…」をクリックしてください。

□「ツール」の前の□をクリックしてチェックを入れてください。

□ワールドをクリックしてワールドメニューを出してください。

□「フラップ…」をクリックしてください。

□「スクイーク」の前の□をクリックしてチェックを入れてください。

これで出来上がりです。

□下の行をクリックしてしてください。

http://www.languagegame.org:8080/zoo/53

みなさん実際に試しながら読み進んでくださいね。

ここまでのまとめ

「条件文」当たりから少しずつ難しくなってきて、大変そうだと思った人も Thoru Yamamoto さんの親切なサイトを見て色々できるようになったかと思います。これでみなさん基本はすべて習得しました。

復習を兼ねて以下のウェブページを見てみてください。

http://www.h3.dion.ne.jp/~y.ich/Squeak/squeak-qref.html

ウェブを徘徊するロボット

では、いよいよクラスを作るプログラミングに挑戦しましょう。

クラスを作るぐらいになると、いきなり作り始める前に、テーマや予備実験が必要にな

ります。まず、テーマの検討から。

ウェブの構造

「ウェブには、誰も訪れたことのない、ロボットでさえも見たことのない大陸が存在するのである。」

A．バラバシ『新ネットワーク思考』

みなさんはウェブを見るのが好きですか？あっちこっちクリックすると、どんどんページが表示されて、いくらでも情報が出てきます。クリックする場所にはたいてい関連する単語が書いてあるので、みなさんの興味のある情報が見つけやすくなっています。

でもたまには、リンクに任せて飛んでいくのも面白いかも。あなたのお気に入りのページから始めて、ページのリンク先の中からランダムに選んで飛んでいくというルールで、一体どこのサイトへ行くのか遊んでみませんか？

もちろん自分でブラウザを使って遊ぶこともできるのですが、どうせなら勝手にあちこち飛んでいってくれるロボットをスクイークで作りましょう。

プログラミングを始める前にちょっとだけウェブについて寄り道を。ここをクリックしてそのページの下のほうにある Figure9 を見てください。

これは、2000 年に AltaVista や IBM や Compaq の研究者が発表した、ウェブの中の 2 億ページを分類して見つけた構造です。

中央の SCC は、リンクをたどると互いに行き着くことができるページの巨大な集まりです。右の OUT は、SCC からたどり着けるが SCC へはたどり着けないページです。リンクの張っていないページとかここに含まれます。左の IN は、SCC や OUT にはたどり着けるけれども SCC からたどり着けないページです。既存のページにリンクを張った新しいページや他から興味を持たれずリンクが張られないページがここに含まれます。その他に、SCC にたどり着けない「半島」や「島」や「トンネル」が存在しています。これに対して、SCC や IN, OUT は大陸と呼ばれています。

検索エンジンが有名なサイトからリンクをたどってページを集める場合、検索エンジンには SCC と OUT しか見えないことになります。IN や半島や島やトンネルにたどり着くリンクがないからです。

2 億ページを調べたところ、半分以上は検索エンジンから見えないページだったそうです。

実際のウェブでは、検索エンジンから見えるページの比はもっと少ないことになるようです。というのは、新しいページが IN や半島や島に付け足されるのと平行して、SCC や OUT のページが更新されて SCC や OUT 自体の数がどんどん増えているからです。増え方

が検索エンジンの収集スピードよりも速いと SCC や OUT の中に検索エンジンがたどり着いていないページが増えていきます。

ちょっとびっくりしませんでした?

ではウェブ大陸を渡り歩くロボットを作りましょう。

試してみよう

（注）このページはワークスペース変数を使いたいのでワークスペースに丸ごとコピーして読んでください。

スクイークには内蔵ソフトのソースコードがすべて入っています。ということは、みなさんは Scamper だけでなく、Scamper が使っているクラスも直接使うことができるんです。

早速使ってみましょう。

クラス String（文字列）のインスタンスに asUrl とお願いすると、スクイークは文字列を URL(Uniform Resoruce Locater)として理解してくれます。URL というのはみなさんもうお馴染みですね。インターネット上の情報の場所を示すアドレスのことです。スクイークが URL を理解してくれているので、その情報を引き出して、とお願いすることができます。

```
url _ 'http://squeak.hp.infoseek.co.jp/' asUrl.
page _ url retrieveContents.
```

上の式を評価してインスペクトするとクラス MIMEDocument のインスタンスだということがわかります。MIME は Multipurpose Internet Mail Extensions の略で、例えばメールで日本語など色々な言葉の文字を送ったり添付ファイルを送ったりする時に使われる汎用的な規格です。インスペクトウィンドウでインスタンス変数を見ていくと、subType に html という情報がありました。HTML はみなさんお馴染み、ウェブページで使われている文書フォーマットの規格です。

さて、このページから次のページへ適当に飛ぶには、ページの中のリンクを集める必要があります。HTML の文には、内容を示すテキストやリンク情報などを示すタグが混ざっていますが、スクイークは既にこれらの情報を整理、解剖してくれる機能を持っています。Scamper のようなウェブブラウザを作ろうとすると必ず必要になる機能ですから。こんな感じにスクイークにお願いします。

```
html _ HtmlParser parse: page content.
```

どうやって asUrl とか retrieveContents とか HtmlParser とか見つけるのかって？それはおいおい説明しますが、マニュアルとかがなくても、右クリックメニューの中の browse it メニューや implementers of it メニューなどでそれらしい単語を検索すると見つかるものなのです。スクイーク(Smalltalk)の世界では、（他の世界もそうですが）できるだけ他の人が探しやすい意味のわかりやすい名前をクラスやメソッドに付けるように奨励されています。

さて、変数 html をインスペクトすると、これはクラス HtmlDocument のインスタンスだということがわかります。インスタンス変数 contents にクラス OrderedCollection のインスタンスとして解剖結果が入っています。OrderedCollection は「要素を順序立てて集めたもの」という意味です。（クラス OrderedCollection はとても便利なクラスなので、みなさんがプログラミングが得意になったらどんどん使うことになるでしょう。）

今度は、変数 html をエクスプロアして、左の三角マークをいくつかクリックしてみてください。インスタンス変数 contents は OrderedCollection が入れ子になっていることがわかります。つまり要素の 1 つが OrderedCollection のインスタンスだったりします。

三角マークをクリックしていくと、リンクを示す'a' href=http://www.... という行が見つかるかと思います。その行を選択して右クリックメニューから検査（インスペクトのことです）を実行すると、この行がクラス HtmlAnchor のインスタンスだということがわかります。この中の href の値がリンク先の URL です。

今は、みなさんがエクスプロアウィンドウを使ってアンカ(Anchor)タグを見つけましたが、スクイークがタグをみつけて処理するには、html の要素 1 つ 1 つを繰り返し調べるコードを書く必要があります。1 つ 1 つ繰り返すことを列挙する(enumerate)と言いますが、クラス HtmlDocument のような「集まり」を含むクラスには、enumeration（列挙）のカテゴリが定義されているはずです。ありゃ？ないですね。そういう場合は、階層ブラウザで見てみてみましょう。スーパークラスに定義されているかもしれません。ありましたありました。HtmlDocument のスーパークラス HtmlEntity にカテゴリ enumeration があって、その中にメッセージ allSubentitiesDo:が定義されていました。all subentities do: aBlock はすべてのサブエンティティについて aBlock を実行するという意味なので、入れ子のエンティティも含めて調べることができそうです。辞書で単語の意味を調べて機能を推測することがスクイークではとても大切です。

Transcript にこのページにあるリンクをすべて表示してみましょう。まず、ワールドメニューから Transcript を開きます。それから、以下の式を実行します。

```
html allSubentitiesDo:
    [ :each |
```

```
(each isMemberOf: HtmlAnchor) ifTrue: [Transcript show: (each attributes at: 'href'); cr]].
```

上の式を評価するには 3 行を選択して「式を評価」を実行してください。

ありゃ私はエラーが出てしまいました。みなさんはどうですか？

HtmlAttributes(Dictionary)でキーが見つからなかったと言っているようです。どうやら href というキーが見つからなかったようです。クラス Dictionary をブラウザで少し調べてみましょう。メソッド at:の次に at:ifAbsent:というメソッドが見つかりました。"if absent" は「もし居なかったら」という意味なので、この後にキーが見つからなかった時の処理をブロックとして渡せばよさそうです。

```
html allSubentitiesDo:
    [ :each |
    (each isMemberOf: HtmlAnchor) ifTrue:
        [Transcript show: (each attributes at: 'href' ifAbsent: []); cr]].
```

さて、うまくリンク先 URL を表示することができましたか？

メソッド isMemberOf:はレシーバの引数のクラスのインスタンスかどうかを返します。

attributes は HtmlAnchor のインスタンス変数 attributes を返します。

attirbutes はクラス Dictionary のインスタンスなので、at:ifAbsent:は引数のキーがもしあればキーに対応する値を返します。なければ ifAbsent:の後のブロックを実行します。今は空のブロックを渡しているので見つからなかったら何もしません。

（クラス Dictionary もとても便利なクラスの 1 つです。これはオブジェクトとオブジェクトのペアを複数保持してくれます。単語と意味のペアを集めると文字通り辞書(Dictionary)になります。）

これでウエブページのリンク先を抽出することができそうです。

メソッドに組み込もう

さて、リンクを抽出する機能をスクイークに組み込みましょう。

組み込む前に、最初にどこに組み込むか考えます。前の節で試した式は html にお願いする形になっているので、クラス HtmlDocument のメソッドとして組み込むのがよさそうです。名前は headAnchors にしましょう。専門用語になってしまいますが、リンクはある場所からある場所に張ることでできるのですが、出発場所をテイル（尾）アンカ、到着場所をヘッド（頭）アンカと呼ぶからです。

前の節のようにTranscriptに表示するだけではスクイーク自身には扱いにくいので、表示する代わりにURLの集合を返すことにします。メソッドの名前を複数形にすることで、戻り値が集合であることを表しています。

ではHtmlDocumentに以下のメソッドを追加します。

```
headAnchors
	"returns a set of URLs as String."
	| urls |
	urls _ Set new.
	self body  allSubentitiesDo:
		[:each |
		(each isMemberOf: HtmlAnchor) ifTrue:
			[| address |
			address _ each attributes at: 'href' ifAbsent: [nil].
			address ifNotNil: [urls add: address]]].
	^ urls.
```

最初に戻り値となるクラスSetのインスタンスを生成しています。クラスSetは同じ要素を重複して持たない、いわゆる数学でてきた集合です。従って、ページの中に同じ場所へのリンクが複数存在していて、add:で複数回加えても、Setを使うとスクイークは1つと見なしてくれます。

HTMLの内容は大きく分けて、ヘッドと呼ばれる部分とボディと呼ばれる部分に分かれています。ヘッドにはそのページのタイトルや後で出てくるベースタグが含まれます。ボディにはそのページの内容が含まれていて、アンカタグも現れます。アンカタグはボディにだけ現れるので、headAnchorではボディだけ調べるようにしました。

アンカタグの中にhref属性が見つからない時は変数addressがnilを示すようにします。nilというのは「無」を表すちょっと変わったオブジェクトです。hrefが見つからなかった時には「無」にするということです。（スクイークでプログラムをデバッグしていると、よくnilに出会うかもしれません。スクイークは何かを割り当てる前の変数にnilを割り当てるので、例えばインスタンス変数の初期化を忘れたりすると、オブジェクトnilにメッセージを送って、エラーウィンドウが現れたりします。いい加減な値のまま動いて、暴走するということがないのでデバッグしやすい工夫になっています。）

addressはnilでない時だけ、urlsにaddressを加えています。

メソッドの追加の方法は覚えていますか？システムブラウザを使って、クラス

HtmlDocument を選択してからコードペインに入力（ドラッグで上のコードを選択して Alt-c でコピー、Alt-v でペーストできますよ）して、右クリックメニューから「了解」を選択します。

手で入力した場合、タイプミスなどからエラーメッセージが出るかもしれません。つづり間違いとか、かっこの数が合っているか、ピリオドを忘れていないかなどをチェックしてください。

ではうまく動くか早速試してみましょう。

```
|url html|
url _ 'http://www.squeakland.org/' asUrl.
html _ HtmlParser parse: url retrieveContents content.
html headAnchors explore.
```

URL の集合が得られましたか？

うまく行っていそうですが、http://から始まらない URL が多いですね。こういう URL を相対 URL と言います。相対 URL は通常、現在のページの URL との関係で絶対的な位置を決めます。例えば、上の例では、現在のページは http://www.squeakland.org/だから、kids/kidshome.html は、http://www.squeakland.org/kids/kidshome.html を示しています

相対 URL を絶対 URL に変換するように、headAnchors を変更したメソッドを作りましょう。後で、ウェブを飛び回ることも考慮して、HTTP プロトコルを使う URL のみ集めるようにします。（例えば、mailto:も URL ですが、メールは出せても、相手から情報をもらうのはむりですから ...）

クラス HtmlDocument のインスタンスは、そのページの中身を表していて、URL に関する情報は持っていないので、引数としてページの URL をもらうようにします。

```
httpAnchors: baseUrl
    "returns a set of URLs."
    | urls |
    urls _ Set new.
    self body  allSubentitiesDo:
        [ :each |
        (each isMemberOf: HtmlAnchor) ifTrue:
            [| address |
            address _ each attributes at: 'href' ifAbsent: [nil].
```

```
                address ifNotNil:
                        [(address includes: $:) "絶対URLかどうかを調べている。かっこ悪いから直したい"
                                ifTrue: [(address beginsWith: 'http:') ifTrue: [urls add: address asUrl]]
                                ifFalse:        [urls add: (address asUrlRelativeTo: baseUrl)]]]].
        ^ urls.
```

イタリック（斜めのフォント）は前の同じコードです。

普通、引数にはどんなオブジェクトを渡すのかクラスがわかるような名前を付けます。名前を見るとクラスUrlのインスタンスを渡すことがわかります。baseは引数の意味をしています。なぜbaseかは少し待ってくださいね。

絶対URLだった場合、http:で始まるURLを追加するようにしています。

相対URLを扱うのに、クラスUrlに目的にぴったりなメソッドasUrlRelativeTo:があったので使ってみました。as url relative toなので引数に対して相対的なURLとしてという意味になります。

早速試してみたくなるのですが、その前にちょっとおまじないを。クラスUrlに=とhashの2つのメソッドを追加します。

```
= aUrl
        ^ self asString = aUrl asString.

hash
        ^ self asString hash.
```

これらについては後で説明しますね。

さあ使ってみましょう。

```
|url html|
url _ 'http://www.squeakland.org/' asUrl.
html _ HtmlParser parse: url retrieveContents content.
(html httpAnchors: url) explore.
```

いい感じになってきました。

（注）上のように headAnchors や httpAnchors:（Smalltalk を使う人たちは引数を持つメソッド名は XX コロンと「コロン」をはっきり発音するそうです）を定義すると、似たようなコードがあちこちのメソッドに現れることになります。こうなると、同じコードの部分で不具合が見つかると、あちこちを直さないといけません。また、不具合はなくてももっと効率のいいコードを思いついた時とかも同じようにあちこち直さないといけなくなります。こうならないようにするにはどうしたらよいかまたみなさんと一緒に考えたいです。これはまたの機会に。

「ウエブ蝶々」

リンク情報を取り出すことができるようになったので、いよいよ最初の目的だった「ウエブを徘徊するロボット」を作ります。

蝶々が花の間を飛び回る様子に似ているかと思ったので、WebButterfly と名付けてみました。

```
Object subclass: #WebButterfly
    instanceVariableNames: 'history random '
    classVariableNames: ''
    poolDictionaries: ''
    category: 'Network-Web Crawler'
```

システムブラウザのコードペインに上の式を入力して、「了解」するとクラスカテゴリに Network-Web Crawler ができて、その中にクラス WebButterfly が現れます。

このように、スーパークラスにメッセージ subclass:instanceVariableNames:classVariableNames:poolDictionaries:category: を送るとサブクラスを作ってくれます。

instanceVariableNames は直訳すると、「インスタンス変数名たち」です。一体なんだろうと思われるかもしれませんが、クラスから作ったインスタンスそれぞれが持つ変数のことです。

はじめて新しくクラスを作ってみました。簡単でしたね。

インスタンス変数 history にはどこを飛び回ったか履歴を残す予定です。random はアトランダムに飛ぶ時の乱数を計算するためのインスタンス変数です。

どうして新しいクラスを作ったのか少しだけ考えてみましょう。headAnchors の時には、

Workspaceで試してみた時からなんとなくわかったような気がしたかと思いますが、HtmlDocumentに関する処理でした。あるHtmlDocumentの中のリンク情報を集めたい、だからHtmlDocumentにそれをお願いできるようにメソッドを追加したのです。今の場合、ウェブを飛び回りたいのですが、ウェブを飛び回るのは誰にお願いするのがよいでしょうか？Urlとか HtmlDocumentとかにお願いするのは変な感じです。お願いする適切なクラスが思いつかない場合には、新しいクラスを用意します。

実際に飛び回るメソッドをクラスWebButterflyに追加しましょう。

```
flyFrom: aUrl
    | current links |
    random _ Random new.
    history _ OrderedCollection new.
    links _ Set new.
    links add: aUrl.

    [links isEmpty] whileFalse:
        [current _ links atRandom: random.
        Transcript show: current; cr.
        history addLast: current.
        links _ (HtmlParser parse: current retrieveContents content) httpAnchors: current].
    ^ history.
```

最初に乱数生成器randomを用意します。randomはお願いすると乱数を返してくれるオブジェクトです。

historyは訪れた順番がわかるようにOrderedCollectionにします。

linksには訪れたページにあるリンク先の集合が割り当てられる予定ですが、最初に引数のUrlを1つだけ入れておけば、ランダムに選んでもこのUrlが選ばれるので引数のUrlからすたーとすることができます。

変数currentには訪れているURLが入ります。です。linksが空じゃなかったら、アトランダムに1つ選んで、historyに追加して、そのページのリンク先を調べます。これをぐるぐる繰り返す。

最初にlinksを引数aUrlだけの集合にすることで、aUrlから調べることになります。

linksが空でない限り、

　linksからランダムにURLを選ぶ。

Transcript に表示する。

history のお尻に追加する。

選んだ URL からページ情報を取ってきてリンク情報を抽出する。

を繰り返し続けます。

クラス Set にメソッド atRandom:という今回の目的にぴったりのメソッドがあったので使ってみました。

さあ、動くか実験してみましょう。Transcript を開いてから、以下の 3 行を評価してください。

```
| wb |
wb _ WebButterfly new.
wb flyFrom: 'http://www.squeakland.org/' asUrl.
```

うまく飛び回りましたか?

スクイーク E トイに比べると、だいぶ地味ですが、面白いことができるものでしょう?

一体、平均何回ぐらい飛び回って止まるものなんでしょうね? 同じところから何度もスタートして、止まるまでの URL の数を求めて平均を計算するプログラムを書いてみてください。

等号

httpAnchors:を作った時、なぜかクラス Url におまじないの=と hash を追加しました。

httpAnchors:ではクラス Set のインスタンスにクラス Url のインスタンスを集めています。クラス Set（集合）は同じインスタンスは何度加えても 1 つだけ保存してくれます。つまり同じか違うかがとても重要になります。ところで、スクイークでは同じかどうかを判断するメソッドが==と=の 2 種類用意されています。

==は ProtoObject と呼ばれるなんだかすごそうなクラスで定義されています。これはオブジェクトが同一のものかどうかを判断してくれます。例えば、

```
a _ 'Hello'.
b _ a.
a == b
```

とすると、変数 a も b も同じオブジェクトを示しているので、a == b の戻り値は true になります。

ところが、

a _ 'Hello'.

b _ 'Hello'.

a == b

とすると、中身は一緒なのですが、'Hello'という文字列インスタンスがスクイークのメモリの別の場所に作られるので、a == b の戻り値は false になります。

a の'Hello'と b の'Hello'が違うと言われると色々困ったことが起こりますよね。そのために別の比較用メソッド=がクラス毎に定義されています。

a = b.

を評価して表示してみてください。true になります。クラス String の場合、中身を比較して同じだったら=は true を返します。

メソッド=が定義されていないクラスでは、=は==と同一になります。

クラス Set では、追加する要素が既にあるかどうか判断する際に、メソッド=を使用します。httpAnchors:ではクラス Url のインスタンスを加えていったのですが、クラス Url にはメソッド=の定義がなかったのです。そうすると、==が代わりに使われるので、a と b の'Hello'が違うと判断されたのと同様に、同じ URL でも別々にインスタンスを作ったからという理由で違うと判断されることになります。これを避けるために、メソッド=を定義しました。定義の仕方は String と一緒にしました。インスタンスを String にしてから比較しています。

ちなみに

a _ 1.

b _ 1.

a == b.

とすると、a == b の戻り値は true になります。文字列と違って、1 のような数字はスクイークには 1 つしかないようです。なんとなくややこしいですね...

ハッシュ関数

追加したメソッド hash の事情はもう少し複雑です。

みなさんは、もしたくさん集めたデータの中から特定のものがあるか探す場合、どんなプログラムを書きますか？ 1 つ 1 つ順に調べると最悪データの数だけ探すのに時間がかかってしまいます。

そこで頭のいい人がハッシュ関数というものを考え出しました。この関数はデータから適当な整数値を計算します。データが同じなら常に同じ値を返し、また異なるデータに対しては値がばらつくことが特徴です。

さて、集めたデータ１つ１つに対してハッシュ関数で整数値を計算します。その整数値１つ１つにデータを収納する配列を用意します。こうしておけば、あるデータがその中にあるかどうか調べるには、まずそのデータのハッシュ値を計算して、その結果に対応する配列だけを調べたらいいことになります。

それぞれのデータのハッシュ値が広く散らばっていれば、調べるデータの量がとても少なくなることが想像できますね。

クラス Set はこの仕組みを使っています。従って、データはハッシュ値を計算できる必要があるのです。ところが=の時と同様にクラス Url にはメソッド hash が定義されていませんでした。この場合、=の時と似ていて、クラス Object のメソッド hash が呼び出されます。クラス Object のメソッド hash は、インスタンスそれぞれを区別する関数になっていて、中身が一緒でも同一でなければ値が異なってしまうのです。それで、=の時と同様に hash もクラス Url に追加しました。定義の仕方はクラス String と一緒にしました。

パーサを変更する

（この章は少し説明が足らなくて読むのが大変かもしれません）

相対 URL は、通常は現在のページの URL を基準とする URL ですが、HTML ファイルの中にベースタグと呼ばれるタグがある場合にはそのタグに記述されているベース URL に対する URL となります。（実はこれを考えていて、httpAnchros:の引数を baseUrl と名付けたのです。）

```
<base href="http://www.htmq.com/html/base.htm" target="_self">
```

ベースタグはこんな感じのものです。

ところがスクイークでは（何故か）ベースタグをアンカタグのように理解してくれません。ベースタグを持つページをパースすると、コメントとして扱われます。実際に見てみましょう。

```
(HtmlParser parse: 'http://www.htmq.com/html/base.shtml' asUrl retrieveContents content)
explore.
```

スクイークは何でも変更できるプログラミング環境。理解してくれないなら理解してくれるように変更しましょう。

まず、ブラウザでクラス HtmlParser を検索します。あらメソッドがひとつもありません。確か parse:はクラスメソッドでした。クラスペインの下の class をクリックしてください。クラスメソッドを見ることができるようになりましたね。（元に戻すには instance を押します。そのクラスのコメントと見たかったら?をクリックします。）

メソッド parse:を見ると、

```
parse: aStream
	^self parseTokens: (HtmlTokenizer on: aStream)
```

どうやらHtmlTokenizerに何かしてもらってから、メソッドparseTokens:で処理をしているようです。Tokenという単語が2回出てきます。トークンというのは一言で説明するのが難しいのですが、この場合、HtmlTokenzierはテキストデータをHTMLのタグ（<>でくくられた部分）とそれと以外のテキストを識別して並べてくれます。parseTokens:はタグ同士の関係を解釈して、例えば、

```
<a href="XXX">link</a>
```

の場合、linkというテキストにアンカタグ属性を付けたデータを生成してくれます。

どうやらparseTokens:が調べるべきところのようです。見てみると、・・・長いです。大筋を理解するために以下に詳細を...で省略した版を書きます。

```
parseTokens: tokenStream
	| entityStack document head token matchesAnything entity body |

	entityStack _ OrderedCollection new. "以下、パースする前の準備"

	...

	"go through the tokens, one by one"
	[ token _ tokenStream next.  token = nil ] whileFalse: [
		(token isTag and: [ token isNegated ]) ifTrue: [
			"a negated token" "閉じタグの処理"

		...]
		ifFalse: [
			"not a negated token.  it makes its own entity" "閉じタグ以外の処理"

			token isComment ifTrue: [ "コメントタグの処理"

			...].
			token isText ifTrue: [ "テキスト（タグ以外）の処理"

			...].
			token isTag ifTrue: [ "タグの処理"

				entity _ token entityFor.
				entity = nil ifTrue: [ entity _ HtmlCommentEntity new
initializeWithText: token source ] ].
```

```
                "entity の処理"
                ...
        ]].

    "後処理"
    ...
    ^document
```

じっと眺めると、トークンがタグの場合、token entityFor の戻り値 entity がキーのようです。entityFor を実装しているクラスを探しましょう。entityFor を選択して、右クリックメニューから「さらに ... 」を選んで、"implementors of it" をクリックします。

HtmlComment, HtmlTag, HtmlToken ですが、それぞれの中身を見ると、HtmlTag が本命のようです。

```
entityFor
    "return an empty entity corresponding to this tag"
    | eClass |
    eClass _ self class entityClasses at: name ifAbsent: [ ^nil ].
    ^eClass forTag: self
```

name はインスタンス変数なのですが、クラス HtmlTag の entityClasses から name に対応する何かを探しているようです。クラスメソッドの entityClasses を見ると、名前とクラス名が並んでいるではありませんか。なんとなく希望が見えてきました。どうやらベースタグに対応するクラスを定義して、ここに追加すればよさそうです。

entityClasses がクラス変数 EntityClasses をクラス Dictionary のインスタンスとして初期化しているのですが、この中で、

```
        title       HtmlTitle
        style   HtmlStyle
        meta   HtmlMeta
```

と、ヘッドの中に含まれるタグが並んでいるところがあります。まず、ここに

```
        base   HtmlBase
```

を追加しましょう。

```
HtmlTag initialize.
```

を評価すると、クラス変数 EntityClasses を再初期化して entityFor が base を意識してくれるようになります。

それから、クラス HtmlBase を追加しましょう。ベースタグは HTML のヘッドと呼ばれる部分にあるので、HtmlHeadEntity のサブクラスにします。

```
HtmlHeadEntity subclass: #HtmlBase
	instanceVariableNames: ''
	classVariableNames: ''
	poolDictionaries: ''
	category: 'Network-HTML Parser Entities'
```

クラス HtmlTitle, HtmlStyle, HtmlMeta をまねて、メソッドを 2 つほど追加しておきます。

```
mayContain: anEntity
	"ベースタグは閉じタグを持たないので何も含むことはありません"
	^ false

tagName
	^ 'base'
```

これだけで、HtmlParser はベースタグを認識してくれるようになります。確認してみてください。

```
(HtmlParser parse: 'http://www.htmq.com/html/base.shtml' asUrl retrieveContents content) explore.
```

スクイークがベースタグを認識してくれるようになったので、WebButterfly>>flyFrom:で、httpAnchors: current という式の前にベースタグを探してベース URL を見つけておけば、current の代わりに正しい baseUrl を渡すことができるようになります。

この変更に取り組む前に、ベース URL をすぐ取り出せるように準備しましょう。

クラス HtmlDocument のインスタンスははクラス HtmlHead のインスタンスとクラス HtmlBody のインスタンスから構成されています。ベースタグは HtmlHead に含まれているはずなので、HtmlHead にベースタグを返すメソッドを追加します。

```
baseEntity
```

```
^ contents detect: [:each | each tagName = 'base'] ifNone: [nil].
```

detect:ifNone:は１つめのブロックがtrueを返す最初の要素を返してくれます。

次に、クラスHtmlDocumentにベースURLを返すメソッドを追加しましょう。

```
baseUrl
    | base |
    base _ self head baseEntity.
    ^ base ifNotNil: [(base attributes at: 'href' ifAbsent: [^ nil]) asUrl].
```

baseEntityからキーhrefの値をUrlとして返してくれます。

さて、ここからはみなさんへの問題にします。

あるURLのページからリンク先の絶対URLの集合を抽出したいです。このメソッドは、そのページのベースタグを見つけて、ベースタグがあればそこから相対URLを絶対URLに変換してくれます。ベースタグがなければ自身のURLから相対URLを絶対URLに変換してくれます。

１．このメソッドをどのクラスに実装しますか？

２．今までに定義したメソッドを最大限に利用して実装してください。

３．このメソッドを使って、WebButterflyを改良してください。

プロジェクト爛柯

クラスも作れるようになったので、いよいよアプリケーションっぽいものに挑戦します。

１９９７年に、ＩＢＭのDeep Blueと名付けられたチェス専用のコンピュータが、チェスの世界チャンピオン、ガルリ・カスパロフさんを6番勝負で勝利しました。センセーショナルに報道されたのでご記憶の方も多いかと思います。その後もコンピュータ対人の戦いは続いて、今のところは互角のようです。市販のソフトもレーティングだけ見ると並みのアマチュアでは歯が立たないようです。

囲碁、将棋では、専用のコンピュータを開発してという大掛かりな話はあまり聞きません。欧米では人工知能がコンピュータサイエンスの一分野として確立しているのに対して、日本ではあまり人工知能の研究が盛んではなかったからかも知れません。しかし、囲碁、将棋は、それぞれチェスとは違った複雑さ（囲碁は盤の広さ、将棋は終盤の手数の多さ）を持つことから、注目されているようです。

思わせぶりなことを書きましたが、ここではコンピュータ囲碁ではなくて、碁盤ソフト

を作ります。碁盤ソフトなので、囲碁を知っている必要はありません。囲碁のルールを実装する予定ですが、囲碁のルールはとても簡単です。一緒に勉強しましょう。

「爛柯」というのは斧の柄が腐るという意味で、囲碁に夢中になっていたら斧の柄が腐っていたという中国の逸話から、囲碁を示す言葉になりました。

Ranka.cs

上のファイルをインストールして、

GoBoardMorph new openInWorld.

を実行すれば、これから作る碁盤ソフトを試すことができます。

碁盤とは？

みなさん、碁盤は見たことがありますね。１９本×１９本の直線が網目上に書かれているあれです。直線の交点に碁石を置いて使います。ここでは碁盤と碁石とセットで碁盤として考えます。（碁盤だけあってもゲームができないですものね。）

コンピュータの中で、碁盤を表現するにはいくつかの方法があります。

各要素に、{空, 黒, 白}のいずれかを持つ１９×１９の２次元配列を作る。

黒石と白石クラスを用意して、各インスタンスに座標情報を持たせる。このインスタンスの集合を碁盤とする。

他にも色々考えられそうです。どちらがいいか作って色々比較してみないとわかりません。まずは１の方法で作ってみましょう。

２次元配列といえば、スクイークでは Matrix というクラスがありますが（バージョン 3.7 を使っています。それ以前では Array2D というクラスがありました。）充実している反面、どうやら数学の行列を表しているようです。ちょっと違うので、新しく SquareLattice2D というクラスを作りました。

SquareLattice2D

SquareLattice2D をブラウザで見ながら読んでください。

機能について

クラス Point のインスタンスを使って２次元の点にオブジェクトを置くことができます。

少し工夫したのは、メソッド adjacenciesOf:do:です。ある点の上下左右の隣接点の集合を返してくれるのですが、このメソッドを使うことで、境界での特殊性（境界は隣接点が少ない）を気にすることなくプログラムを書くことができるようになります。

実装について

単純なアクセスしかしないので、クラス Array のインスタンスにオブジェクトを保存することにしました。クラス Array を継承することは考えませんでした。Array は 1 次元配列で、SquareLattice2D は 1 次元配列ではないですから。

Matrix が Collection を継承していたので、Collection のサブクラスにしようかどうか迷ったのですが、Collection のプロトコル（プロトコルというのは、プログラマの間の取り決めという意味で、ここではある決まった名前のメソッドたちを示します。）を使うかどうかわからなかったので、Object から継承しました。

最初、subclass:...の代わりに variableSubclass:... を使ってみたのですが、variableSubclass:... で定義されているクラスを列挙してみると、特別なものが多かったので、配列はインスタンス変数として持つことにしました。

Smalltalk allClasses do: [:e | (e isVariable and: [e superclass notNil] and: [e superclass isVariable not]) ifTrue: [Transcript show: e; cr]].

交点の状態

碁盤の交点は、

何もない

黒石がある

白石がある

の 3 つの状態がありえます。

これを表現する一番簡単な方法として、SquareLattice2D の各点に、3 つのシンボル (#empty, #black, #white)のいずれかを持たせることが考えられます。実際にこれで一度実装してみました。そうすると、コードの中に、

(squareLattice2D at: aPoint) = #empty ifTrue: [...].

(squareLattice2D at: aPoint) = #black ifTrue: [...].

(squareLattice2D at: aPoint) = #white ifTrue: [...].

が何度も現れそうです。

例えば、碁盤を表示するとき、

(squareLattice2D at: aPoint) = #empty ifTrue: [Transcript show: ' '].

(squareLattice2D at: aPoint) = #black ifTrue: [Transcript show: 'X'].

(squareLattice2D at: aPoint) = #white ifTrue: [Transcript show: 'O'].

のくり返しという感じです。

このように状態に応じて処理を変える場合、オブジェクト指向言語は美しい方法を提供してくれます。それは State パターンと呼ばれるデザインパターンです。

State パターンでは、状態それぞれに対して、クラスを用意します。具体的には、GoEmpty, GoBlack, GoWhite というクラスを作りました。そして、それぞれについて、メソッド printOn:を用意します。

```
GoEmpty>>printOn: aStream
    ' ' printOn: aStream.

GoBlack>>printOn: aStream
    'X' printOn: aStream.

GoWhite>>printOn: aStream
    'O' printOn: aStream.
```

そして、squareLattice2D の各点に GoEmpty、GoBlack もしくは GoWhite のインスタンスを持たせると、先ほどの表示コードは、

```
Transcript show: (squareLattice2D at: aPoint)
```

の一行で書くことができます。ちょっと魔法のようでしょ？

GoEmpty, GoStone, GoBlack, GoWhite

GoEmpty, GoStone, GoBlack, GoWhite をブラウザで見ながら読んでください。

機能について

表示の際の個別の情報をそれぞれが用意しました。具体的には、テキスト出力の printOn:と後で碁盤のモーフを作る際、色情報を返す color です。

また、よかったかどうか定かではありませんが、アゲハマ（囲って取った石のこと）の管理も GoBlack, GoWhite がするようにしました。

実装について

GoStone は、GoBlack, GoWhite の抽象スーパークラスです。

コンピュータ碁盤

ここで碁盤プログラムを作るために囲碁のルールを説明します。

囲碁では、

相手の石の固まりの縦横を自分の石ですべて囲ったら相手の石を取る。

というルールがあります。

せっかくコンピュータの碁盤を作っているのだから、これを自動で実行してくれる碁盤を作りましょう。これを実装するには、石の固まりというクラスが必要そうです。（実際には、必要かどうかわからないので、最初は先に作った SquareLattice2D と GoEmpty, GoBlack, GoWhite を使って作りました。作っているうちにこれはクラスにしたほうがよさそうだという感じがしてくるのです。）

GoGroup

GoGroup をブラウザで見ながら読んでください。

機能について

石の種類、固まりの位置情報、空いている隣接点情報を持ちます。

実装について

石の固まりは、位置情報の集合なので Set をスーパークラスとしました。それとは別に空いている隣接点の集合をインスタンス変数として持ちます。今のバージョンではこれから説明する GoBoard の groupAt:メソッドが GoGroup の中身を生成するので、GoGroup に中身の生成を委譲するような構造がいいかもしれません。まだうまく表現できていないです。

さあ、碁盤本体を作る準備ができました。

GoBoard

GoGroup をブラウザで見ながら読んでください。

機能について

playAt:で位置を与えると、

自動的に交互に打つ

取るべき石は取る

打ってはいけない場所ではエラーが出る。

実装について

SquareLattice2D のインスタンスをインスタンス変数に持ちます。GoEmpty, GoBlack, GoWhite のインスタンスもそれぞれインスタンス変数として持ち、交点の状態を表します。SquareLattice2D の各点は、先の 3 つのインスタンスのいずれかを示すようにしています。

表示

これで碁盤のコンピュータ上のモデルができました。このモデルを表示するモーフを作ります。このとき重要になるのが、ＭＶＣパターンです。

Smalltalk では、オブジェクトとオブジェクトがやり取りする方法に、メッセージの送信がありますが、その他にディペンデンシーというフレムワークが用意されています。

碁盤のコンピュータモデルを作りましたが、その表示プログラムを書こうとすると、モデルが変更された時それを知る必要があるわけです。ところが変更した時表示メソッドを送信するようなことにはコンピュータモデルは関わりたくないのです。（他の例でいうと、時刻は、表示がアナログ表示かデジタル表示か自分では決めたくないのです。）

こんな時、ディペンデンシーを使います。

ディペンデンシーフレームワークは、インスタンス a にインスタンス b を登録（a addDependent: b）すると、a に changed というメッセージが送信された時、b に update というメッセージを送信してくれます。

GoStoneMorph

機能について

これは、碁石を示すモーフです。黒や白の円を描きます。最初から碁盤に並べられていて、石がないところは透明にします。

碁盤上のマウスイベントはこのモーフが扱います。具体的には、クリックされると、その位置を引数として GoBoard のインスタンスに playAt:を送ります。

実装について

円なので EllipseMorph を継承してました。

GoBoardMorph

機能について

碁盤プログラム本体です。

モデルとモーフを結びつけたり、GoStoneMorph を並べたりという準備をしたり、モデルが更新された時に表示を更新します。

実装について

モデルの管理をしたかったので、MorphicModel を継承しました。

つづく