第14章　Nspireのプログラミング

Nspire には，そのコマンドを組み合わせてプログラミングをする機能があります．たとえば，新しい関数を定義するとき，それを一つの式で表すことができるときは「Define」を利用した定義をするだけで済みますが，その関数が幾つかの複雑な処理を経て計算されるような場合は，その処理をプログラムで作成する必要があります．

Nspire では，TI–89Titanium や voyage200 等の旧機種と同じ TI–BASIC によるプログラミングが可能ですが，残念ながらグラフ描画関係のコマンドはすべて省かれています．つまり，Nspire 本体で TI–BASIC を利用したグラフ描画を含むプログラミング，たとえばゲームの作成などをすることはできません．ただし，Student Software のツールバーの「Insert」に登録されている「Script Editor」を利用すると，プログラミング言語「Lua」を利用することできます．したがって，Student Software の「Script Editor」を利用して Lua 言語で記述すればグラフ描画を含むプログラミングが可能であり，それを Nspire 本体に転送すれば Nspire でグラフ描画を含むプログラムを実行することができます．

この章では Nspire のプログラミング機能について解説しますが,「プログラミング」に関する一般的な知識は所持していることを前提とします．つまり，他のプログラム言語で簡単なプログラムは作成できることを前提とします．そのような知識のない方は，プログラミングに関する他の一般的な入門書を最初に参照してください．

プログラミング言語 Lua と Nspire で利用できる Lua の詳細は，下記 URL を参照してください．

ただし，Script Editor により「Lua」の全機能を利用できるわけではありません．TI–BASIC で除かれたグラフ描画関係を補充するために必要な部分だけが組み込まれているようです．

[1] Lua 5.3 リファレンスマニュアル（非公式・日本語訳）
http://milkpot.sakura.ne.jp/lua/lua53_manual_ja.html
[2] TI-Nspire Lua Scripting API Reference Guide
https://education.ti.com/en/guidebook/search/ti-nspire-cx

40　ライブラリー

Nspire では，一つのドキュメントは 30 のプロブレムを持つことができ，一つのプロブレムは 50 のページを持つことができます．そして，一つのページには 99 の履歴を保持することができます．個々のページで作成した関数やプログラムは同じプロブレム内では共有されますが，他のプロブレムや他のドキュメントでは共有することができません．あるページで定義した関数やプログラムを他のプロブレムやドキュメントで利用できるようにするには，その関数やプログラムを**ライブラリー**に登録する必要があります．

40.1　ライブラリーへの登録

定数の値や関数を定義して他のプロブレムやドキュメントでも利用できるようにするには，次の手順で定義する必要があります．簡単な例として,「3.14」を文字定数「constpi」に定義する場合を考えます．他のプログレムやドキュメントで使用できるようにするには，ドキュメントの最初のプロブレム内のページで定義し，そのドキュメントはフォルダー「MyLib」に保存する必要があります．

(1) ライブラリー登録用の新たなドキュメントをオープンして,「Program Editor」を追加する．
(2) 追加すると，図 40.1(b) のように 3 つの項目の入力を求められる．
　i) 「Name」の箇所では，後で参照するコマンド名を定義する．ここでは,「constpi」とする．
　ii) 「Type」の箇所では，ここで定義するものがプログラム (Program) なのか関数 (Function) なのかを指定する．何らかの一連の処理を行うときは「Program」を，何らかの計算を行って値を返すときは「Function」を選択する．ここでは,「Function」で定義している．
　iii) 「Library Access」の箇所では，ここで定義するものを他のプロブレムやドキュメントで使用するかどうかを指定する．使用しないときは「None」を，使用するときは「LibPrib」を，使用してさらに Catalog にも登録するときは「LibPub」を選択する．つまり，LibPub を選択すると，自分で定義したプログラムや関数を Catalog 表示に追加することができる．ここでは，LibPrib を指定した．

(a) 「Program Editor」のページを追加　　(b) 関数の定義

図 40.1: Program Editor による関数定義

(3) 以上を指定して enter を押すと，図 40.2(a) のような画面が表示される．これを**プログラム画面**という．1 行目には関数の名前が表示され，具体的な定義は 2 行目以降で行う．関数もプログラムも，2 行目は「Define name()=･･･」の形で表示される．LibPrib や LibPub のオプションを指

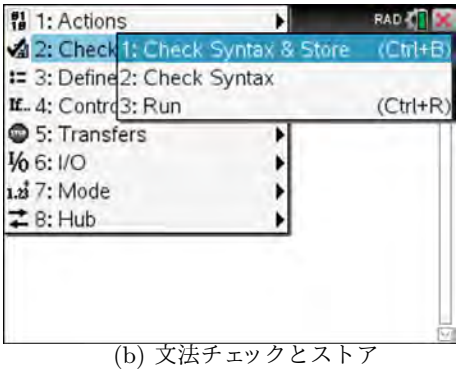
定すると，Define の後に表示される．name() の括弧内は，引数があるときは処理内容により必要なものを指定する．

関数 (function) の場合は，「Func ··· EndFunc」の間に必要な処理を記述する．プログラム (program) では「Prgm ··· EndPrgm」と表示される．いずれにおいても，具体的な記述をするにはプログラミングの知識が必要となる．プログラミングに必要なコマンドは次節で解説する．

こみ入ったプログラムを作成するときは半角英数字を駆使するので，Nspire 本体より Student Software の方で作成した方がよいと思われる．

(4) constpi は数値を表示するだけなので，プログラムの本体には「3.14」を書き込む．必要な記述を終えたら，図 40.2(b) のように menu 2 から「1: Check Syntax & Store」を選択して文法チェックを行う． ctrl B でもかまわない．エラーがなければ，定義内容が Nspire の実行可能な中間言語に変換されて保存され，「”constpi” stored successfully」として保存成功のメッセージが表示される．文法チェックを行うだけであれば「2: Check Syntax」を選択してもよい．エラーがなければ「No syntax erros found」のメッセージが表示され，エラーがあるとカーソルが最初のエラー付近に飛ぶので，必要な修正をする．

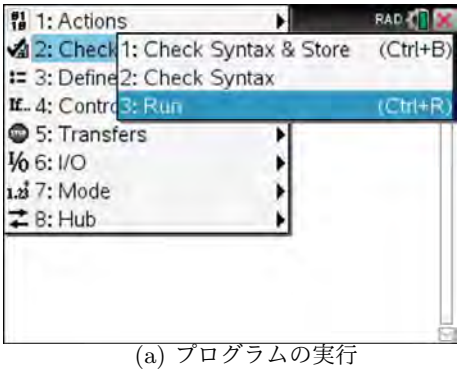
(a) プログラム本体



(b) 文法チェックとストア

図 40.2: プログラム作成と文法チェック

(5) 作成した関数やプログラムが正しく実行できることを確認するには， menu 2 から「3: Run」を選択する． ctrl R でもかまわない．図 40.3(a) のように計算画面が追加されて定義した関数やプログラム名が表示されるので，必要な引数を入力して enter を押す．constpi は引数が不要なので単に enter を押す．引数がない場合であっても，「()」を削除してはいけない．



(a) プログラムの実行

(b) 実行結果

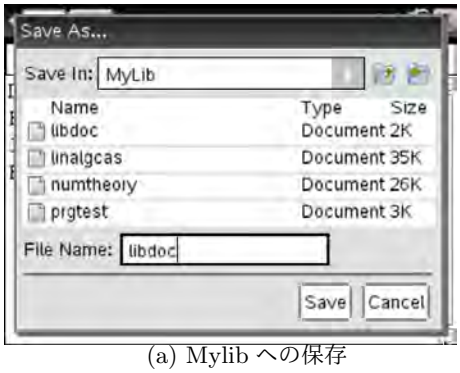
図 40.3: プログラムの実行

(6) 以上のようにして関数やプログラムを作成することができる．しかし，まだ他のプロブレムやドキュメントで実行できる状態ではない．他でも実行できるようにするには，次の 2 つのことを行う必要がある．

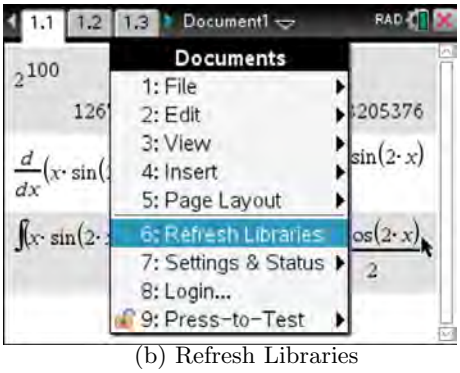
i) 関数やプログラムを定義したドキュメントを,「TI-Nspire CX」のフォルダー「Mylib」の中に保存する．ファイル名を，ここでは「libdoc」とした．ドキュメント名の変更は，[doc] [1] から「5: Save As」により行うことができる．保存フォルダーは，必ず「Mylib」を指定する．

ii) ドキュメント (libdoc) を保存した後では，新しい関数やプログラムをライブラリーに登録しことを Nspire に知らせる必要がある．そのため，[doc] から「6: Refresh Libraries」を選択する．

図 40.4(b) では，別なドキュメントを開いて [doc] [6] を押している.「Refresh Libraries」は，ドキュメントを保存後に一度行うだけで良い．



(a) Mylib への保存



(b) Refresh Libraries

図 40.4: 登録した関数・プログラムの保存とライブラリーの更新

(7) 新たに登録した関数「constpi」は「libprib」を指定しているので，他のプロブレムやドキュメントでも実行することができるが，それは計算画面やノート画面 (Notes) においてである．他のアプリケーションでは実行できない．また,「別なドキュメント」は，フォルダー「Mylib」内のドキュメントである必要はない．他のフォルダーにあるドキュメント内でも実行することができる．ただし，計算画面やノート画面 (Notes) での実行に限られる．

図 40.5 は前章で使用したドキュメントである．すでに [doc] [6] を実行済みであるが，計算画面で「constpi()」としても，右図の下から 2 行目のように値は表示されない.「constpi()」がどのドキュメントで定義されているかを指定する必要がある．この関数は「libdoc」というドキュメントで定義されているので，先頭にそのファイル名をつけて「libdoc\constpi()」として実行する．この追加を行うと，最後の行のように「3.14」が表示される．

記号「\」は，[shift] [÷] により入力する．

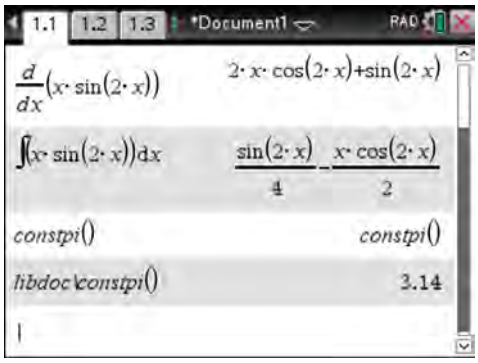


図 40.5: 他のドキュメントでの実行

「Define」だけによる登録

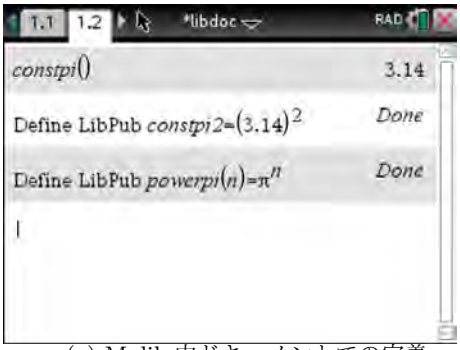
単に「3.14」を表示させるために引数を持たない関数 constpi() を作成しましたが，いろいろな文字定数に値を登録するには，わざわざプログラム画面を利用しなくても直接「Define constpi=3.14」とするのが簡単です．しかし，このような定義は一つのプロブレム内だけの定義であり，他のプロブレムやドキュメントでは利用できません．

プログラム画面を利用しないで，計算画面で直接「Define LibPub constpi2=3.14^2」として定義する方法もあります．この定義の場合,「LibPriv」は指定できません．LibPub を用いた定義なので，カタログにも登録されます．同様の方式で，簡単な式で表される関数を登録することができます．たとえば,「Define LibPub powerpi(n)=π^n」のような定義ができます．ただし，定義式が長くなるときはプログラム画面を利用した方がよいと思います．

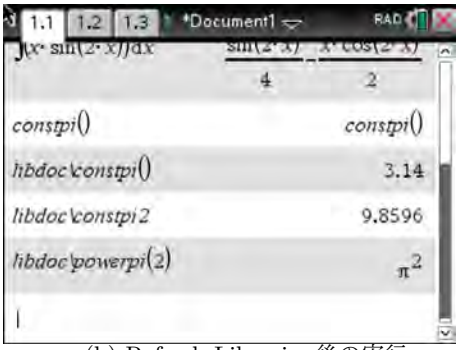
このような形で定義するとしても，次のことに留意する必要があります．

(1) 定義の書かれたページは，ドキュメントの最初のプロブレム内におく．
(2) そのドキュメントは，フォルダー「Mylib」に保存する．
(3) 保存後は，[doc] [6] を実行してライブラリーを更新する．この操作を行わないと，他のドキュメントで利用することはできない．
(4) 定義したコマンドを実行するときは，[shift] [÷] を利用して，コマンドが定義されているドキュメント名を先頭に追加する必要がある．

したがって，いろいろなドキュメントで利用できるような自作の定数・関数・プログラムを作成する場合は，それらを定義するためだけのドキュメントを別に作っておくのがよいと思います．



(a) Mylib 内ドキュメントでの定義



(b) Refresh Libraries 後の実行

図 40.6: 登録した関数・プログラムの保存とライブラリーの更新

図 40.6(a) では，前節に Mylib に保存したドキュメント (libdoc) に，定数 constpi2 と関数 powerpi(n) の定義を追加しています．(b) では，libdoc を保存した後に別なドキュメントを開いて，[doc] [6] を実行してライブラリーを更新してから定義済みの内容を確認しています．

LibPrib 指定をした constpi() は [catal] で表示されるカタログには登録されませんが，他のドキュメントで利用することができます．一方，LibPub 指定をした constpi2 と powerpi(n) はカタログに登録されます．

以上のことを確認してみましょう．カタログに登録されるといっても，Nspire の全てのコマンドをアルファベット順に並べた箇所に登録されるわけではありません．自作のコマンドは,「6:」の箇所に登録されます．Mylib 内のドキュメント名が表示されるので，libdoc の箇所をクリックすると，図 40.7 にあるように，constpi は表示されませんが，constpi2 と powerpi が表示されており，LibPub で指定した 2 つのコマンドがカタログに登録されたことが分かります．他のドキュメントで利用するときは，この画面から指定することもできます．

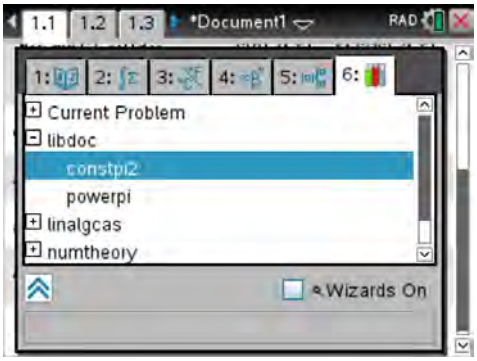


図 40.7: カタログへの登録確認

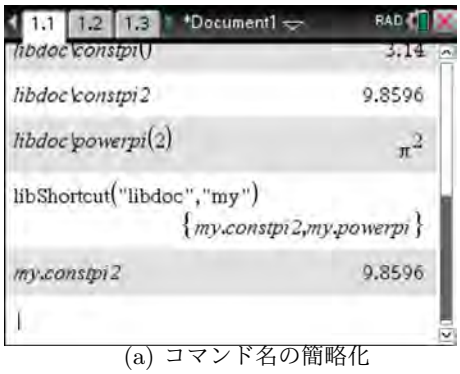
ライブラリーからの削除

ライブラリーから削除するには，LibPrib または LibPub の指定を外す，定義ページを削除する，または定義ページを同じドキュメントの 2 番目以降のプロブレムに移動させる，という 3 つの方法があります．つまり，必ずしも定義ページを削除する必要はありません．変更後は，そのドキュメントを保存して，さらに doc 6 によりライブラリーを更新することを忘れないでください．

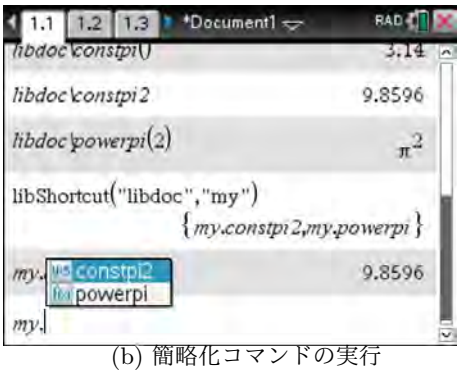
コマンド名の簡略化

自前のコマンドを利用できるにしても，定義したドキュメントの名前を shift ÷ を押しながら付けるのは煩わしすぎます．このとき，入力を簡略化する方法があります．

コマンド等を定義した Mylib 内のドキュメント名を「docname」とし，それをたとえば「dn」と略称したいときは，Nspire のコマンド「libShortcut」を利用して「libShortcut(”docname”, ”dn”)」とします．これにより，ドキュメント docname で定義されている全ての関数やプログラムを「dn.」をつけるだけで参照することができます．必ずピリオド「.」をつけてください．



(a) コマンド名の簡略化



(b) 簡略化コマンドの実行

図 40.8: コマンドの簡略化と実行

たとえば，libdoc で定義した constpi2，powerpi がすでにライブラリーに登録済みです．ドキュメント libdoc を「”my”」と簡略化することにすると，計算画面で図 40.8(a) にあるように，

libShortcut(”libdoc”, ”my”)

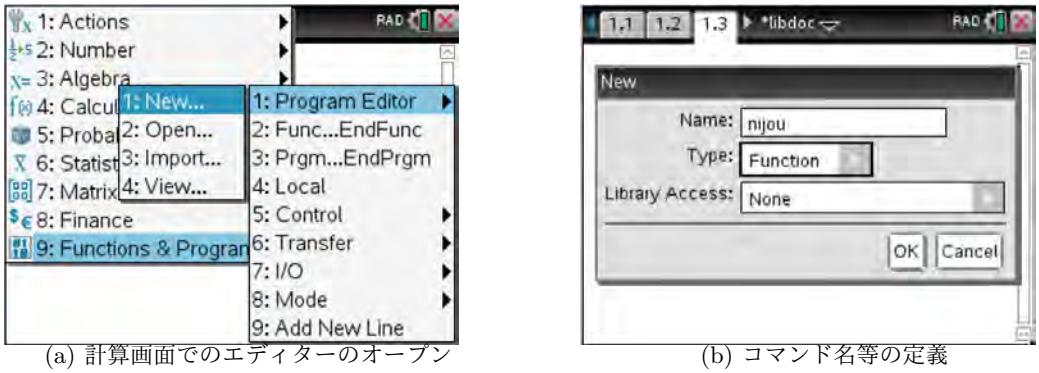
を実行します．そうすると，libdoc 内で定義された関数やプログラムが，すべて「my.」を付すだけ

で参照できることが示されます．ただし，それを実際に利用するには，doc 6 によりライブラリーを一度更新する必要があります．(a) の「my.costpi2」を実行する前に、doc 6 を実行しています．そうすると，(b) のように,「my.」を打ち込むだけで「my.」からはじまるコマンドが表示され，定数なのか関数なのかの区別まで表示されます．したがって，その中から必要とするものを選択するだけで済みます．つまり，libShortcut を利用して簡略化しておくと，コマンドのスペルを入力するのは最初の簡略化されたドキュメント名だけで済むということです．ただし，libShortcut を実行後には，doc 6 によりライブラリーを更新することを忘れないでください．

40.2　計算画面でのプログラミング

前節ではプログラム画面を追加して定義しましたが，計算画面のままでプログラミングを行うこともできます．そのことを試すため，前節で利用した libdoc をオープンして計算画面を追加します．

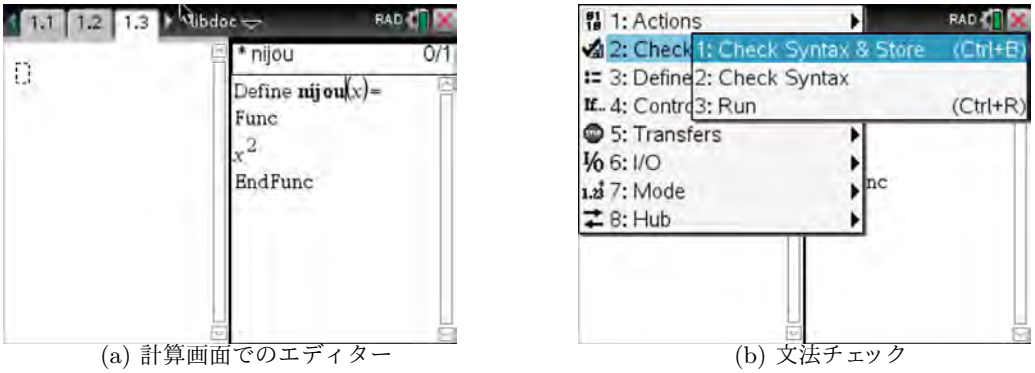
プログラム画面を追加することなく計算画面だけプログラミングをするには，menu 9 から「1: New」を選択します．そうすると，定義しようとしているコマンド名とそのタイプなどが問われます．たとえば，単純に 2 乗するだけの関数 (nijou) を作成してみましょう．関数なので Type は「function」とし，ライブラリーへの登録はしないこととします．図 40.9(b) のように指定します．



(a) 計算画面でのエディターのオープン　(b) コマンド名等の定義

図 40.9: 計算画面でのコマンドの定義

そうすると，図 40.10(a) のように，画面が左右に分割されて右側にプログラム・エディターが表示されます．(a) では，引数を x とし本体に x^2 を書き入れています．プログラムを作成したら menu 2 により文法チェックを行い，問題がなければ「1: Check Syntax & Store」により保存します．



(a) 計算画面でのエディター　(b) 文法チェック

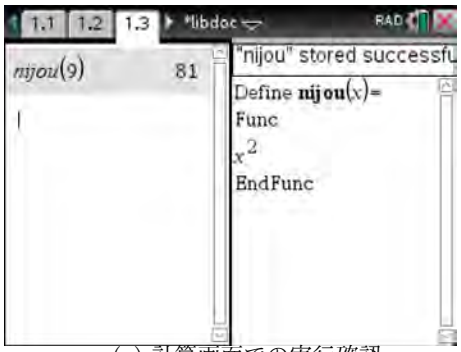
図 40.10: 計算画面でのプログラム作成

ここで，ポインターを左側の計算画面に移すと，図 40.11(a) のように, ページを移動することなく作成したプログラムが正しく実行されるかどうかを確認することができます.

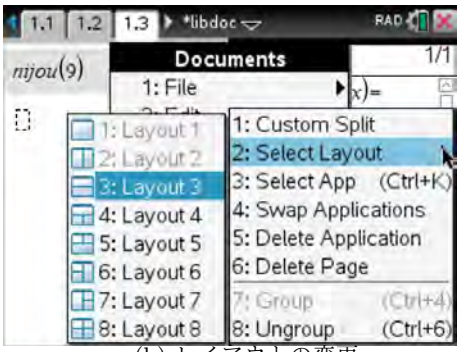
分割画面のレイアウトの変更

左右に分割されていると，長いコマンドを書き入れたときに行替えされて見にくくなります．左右分割ではなく上下 2 段にすることもできます．そのレイアウトの切り替えは，図 40.11(b) のように doc の「5: Page Layout」から「2: Select Layout」を選択します．レイアウトのパターンが表示されるので，そこから希望するレイアウトを選択します．上下 2 段を選択するには doc 5 2 3 を押します．(c) のような上下 2 段の配置になります.

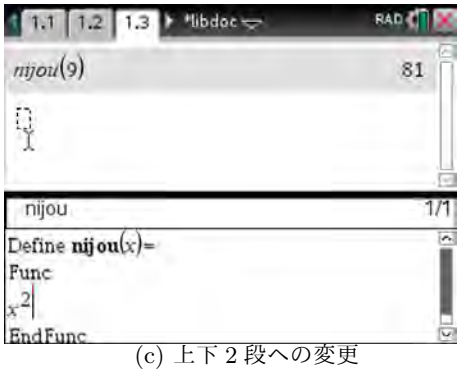
上下や左右の配置を入れ替えるには，(d) のように doc 5 から「4: Swap Application」を選択します．ポインターが置かれている枠が点滅して交換記号が現れるので，その記号を枠を配置したい箇所においてクリックすると，配置を交換することができます．(図は略)



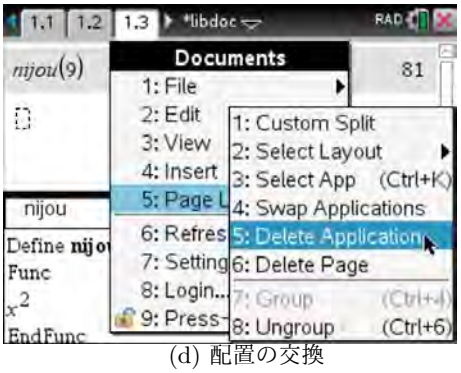
(a) 計算画面での実行確認



(b) レイアウトの変更



(c) 上下 2 段への変更



(d) 配置の交換

図 40.11: 分割画面でのレイアウトの変更

画面分割をやめて計算画面だけに戻るには，ポインターを消去する画面に置いて doc 5 から「5: Delete Application」を選択します．プログラム画面にポインターを置いて doc 5 5 を押すと，プログラム画面が消えて計算画面だけに戻ります．ポインターの置かれている画面が消去されます.「6: Delete Page」を選択すると，画面分割されているページ自体が削除されるので気をつけてください.

プログラム画面を消去しても，そこに書き込んだプログラムまで削除されるわけではありません．文法チェックして保存したプログラムは，画面を消去しても残っています．そのことは， var を押すと確認できます (図は略)．画面が 1 つの状態で doc 5 から「1: Custom Split」を選択すると，画面が左右に分割され右側には Npire のすべてのアプリケーションを指定することができます (図は略).

41　TI–BASIC

この節では，Nspire のプログラム・エディターで作成できる TI–BASIC について解説しますが，プログラミングで使用できるコマンドの種類や使い方等に関して解説します.「プログラミングの仕方」自体については，他のプログラミングの入門書を参照してください.

TI–BASIC は，TI 社のグラフ電卓に備わっている BASIC 言語です．TI–89Titanium や voyage200 などの電卓ではグラフ描画に関するコマンドのみならず，アセンブリ言語や C 言語を用いたプログラミングも可能になっていましたが，Nspire の TI–BASIC ではグラフ描画関係のコマンドがすべて削除されています．グラフ描画関係のコマンドは，Student Software によるスクリプト言語「Lua」を利用することにより可能になっています.

41.1　関数とプログラム

プログラムを作成する上では，関数 (function) とプログラム (program) の違いについて理解しておく必要があります.

関数 (funcion) は一連の計算を行ってその結果を返しますが，プログラム (program) は一連の処理を行うだけであり結果を返すわけではありません．したがって，何らかの数式の中で，すでに定義済みの関数を使用することはできますが，値を返さないプログラムを使用することはできません.

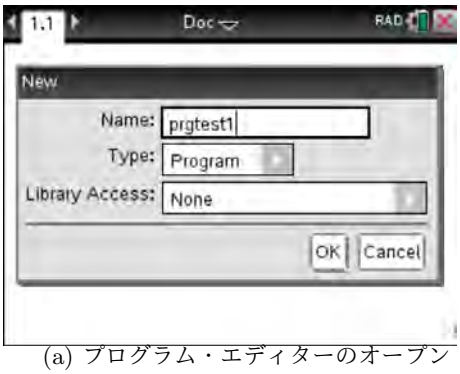
作成したプログラムは，計算画面 (Calculator) とノート画面 (Notes) の中においてだけ実行することができ，グラフ画面やテーブル画面で実行することはできないので注意してください．それに対して，関数は全てのアプリケーションの中で値を評価することができます.

関数を定義するとき，別に定義済みの関数を利用することはできますが，関数の定義の中でプログラムをサブルーチンとして呼ぶことはできません．関数の引数は，自動的にローカル変数として扱われます．プログラムを定義するとき，その中に関数の定義を含めることはできません.

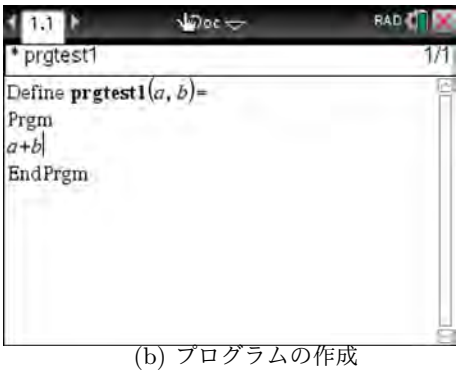
41.2　プログラムの作成

プログラム画面の追加

実際にプログラムを組んでみるため，ドキュメントを新しくして最初のページにプログラム画面を追加します．そこでは，プログラムの名称 (Name)，プログラムか関数かの区別 (Type)，そしてライブラリーへの登録の可否 (Librariy Access) について問われます．とりあえず，名前を prgtest1 として図 41.1(a) のように作成しておきます.



(a) プログラム・エディターのオープン



(b) プログラムの作成

図 41.1: サンプルプログラムの作成

計算画面を左右に分割することでも作成することができますが (40.2 節)，プログラミングを試す上では一つのページとして作成した方が良いと思います．また，ライブラリーへの登録は作成後に変更することができるので，プログラム作成の段階では悩む必要はありません.「None」を指定すると，プログラムが書かれているプロブレム内だけで実行することができます．他のプロブレムやドキュメントでは実行できません.「LibPriv」や「LibPub」を指定すると，他のプロブレムや他のドキュメントでも実行可能になりますが，いろいろな制約があります．40.1 節 (14–5 頁) を参照してください。

画面への表示

簡単なプログラムとして，たとえば 2 数を与えて和を求めるプログラムを作成してみましょう．それは，引数を a, b として単に $a+b$ を返すだけです．図 41.1(b) のようにして menu 2 3 により実行すると，計算画面が増えて「prgtest1()」が表示されますが，たとえば「prgtest1(2, 3)」としても図 41.2(a) の 1 行目のように「Done」が表示されるだけです．これは，和を計算することは指示していても，結果を画面に表示することをプログラム内で指示していないためです.「和を計算した」ということが「Done」として報告されているだけです．

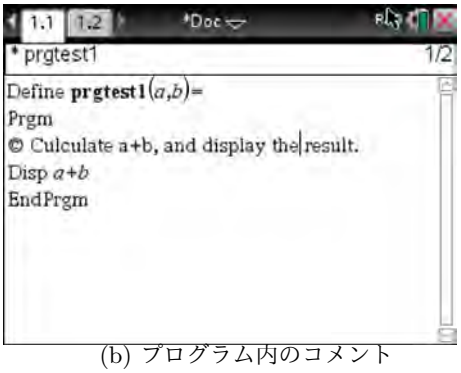
結果を表示させるコマンドは「Disp」です．直接打ち込むか，または menu 6 から「1: Disp」を選択して (図は略)，(b) のように「$a+b$」を「Disp　$a+b$」に変更します．その後に menu 2 3 とすると計算画面に移るので，あらためて「prgtest1(2, 3)」を実行します．今度は，(a) の 2 行目のように結果の「5」が表示されます．menu 2 3 としないで，自分で計算画面に移って「prgtest1(2,3)」を打ち込んでもまいません．

このように，menu 2 1 の「1: Check syntax & store」を経ないで直接 menu 2 3 で「3: Run」を実行してもかまいません．文法エラーがなければ保存されて実行されます．ただし，正しく実行できるプログラムの一部を修正するだけの場合は「3: Run」を選択してもよいですが，ある程度プログラムが長くなるときは，最初に「1: Check syntax & store」か「2: Check syntax」を選択してエラーがないことを確認するようにした方が良いと思います．

コメントの挿入

「Prgm ··· EndPrgm」や「Func ··· EndFunc」の間に，いろいろなプログラム用のコマンドを書き込んでいくことになりますが，何をしているのかが分かるようなコメントを書き入れておくと，後で見たときに分かりやすいです．そのようなコメントは「©」の後に記述します．この記号は，プログラム画面で menu 1 から「8: Insert Comment」を選択することで挿入することができます．(b) では，このコメントが挿入されています．この記号は，ctrl catal にもあります．

(a) プログラムの実行



(b) プログラム内のコメント

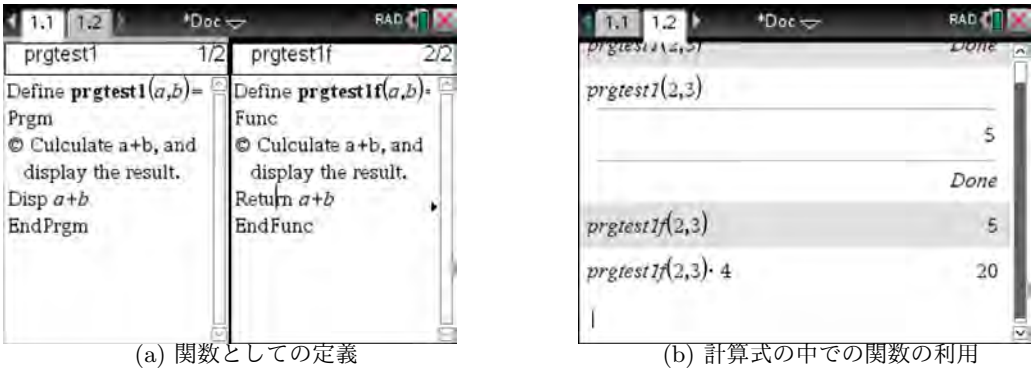
図 41.2: プログラムの実行とコメントの挿入

関数とプログラム

次に，プログラムと関数の違いを見るため，同じ内容を関数として定義してみましょう．

プログラム画面で，menu 1 から「1: New」を選択して，Name を prgtest1f，Type を Function として同じ内容を記述します．図 41.3(a) のように画面が左右に分割されるので，左側の画面を見ながら書き入れるとよいでしょう．画面のレイアウトは doc 5 により変更することができます．プログラムでは「Disp」とした箇所を，関数では「Return」に変更します.「Return」は，定義している関数の値を返すコマンドです．直接打ち込むか，または menu 5 から選択することができます (図は略). 変更後に menu 2 3 として prgtest1f(2,3) を実行すると，(b) のように「5」が表示されます.

関数として定義すると，prgtest1f(2,3) 自体が 5 という値を持ちます．したがって，(b) の最下行のように計算式の中で利用することができます．プログラムである prgtest1 で「prgtest1(2,3)∗4」とすると，prgtest1(2,3) は値を持たないのでエラーが表示されます (図は略).



(a) 関数としての定義　　(b) 計算式の中での関数の利用

図 41.3: 関数と計算式での利用

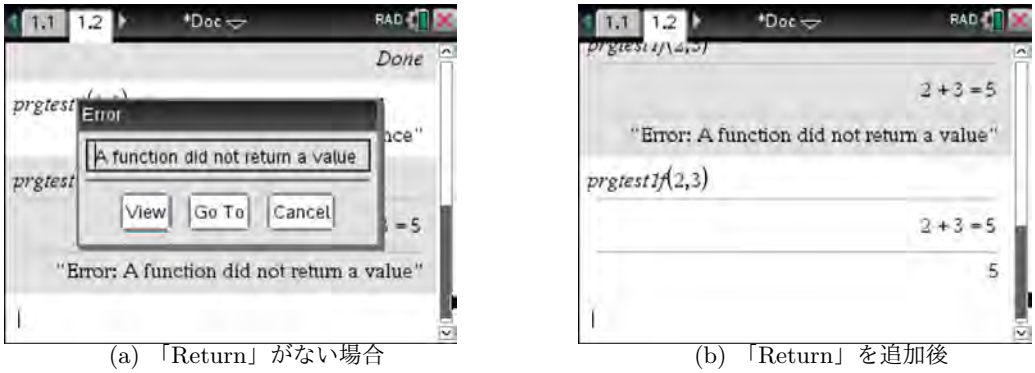
文字の出力

結果を表示するとき，それがどのような内容であるのかが分かるようようにするには，文字を含めながら表示させるとよいでしょう．文字は「” ”」で囲って表示 (Disp) し,「,」で区切ります.

たとえば，プログラム prgtest1 において「Disp $a+b$」の箇所を次のように変更して prgtest1(2,3) を実行すると右側のような結果が表示され，実行後は「Done」が書き込まれます.

プログラム内の記述	実行結果
Disp ”a+b=”, a+b	$a+b=5$
Disp a,”+”,b,”=”,a+b	$2+3=5$

関数 prgtest1f の「Return $a+b$」の箇所を同じ内容で書き変えると，文法的には問題がないので「Check syntax」を行ってもエラーは表示されません．しかし，prgtest1f(2,3) を実行しようとすると，図 41.4(a) のように「A function did not return a value」というエラーが表示されます．このメッセージは，定義している関数の値が「Return」により返されていないことを指摘しています．関数を定義するときは，その関数の最終的な値が何であるかを必ず「Return」を用いて返す必要があるので留意してください．プログラムの最後に「Rturn $a+b$」を追加すると，Disp で指定した表示の後に Return による結果が (b) のように表示されます.



(a) 「Return」がない場合　(b) 「Return」を追加後

図 41.4: 「Return」の有無による関数 prgtest1f の実行結果

値の代入

関数やプログラムの内部でいろいろな計算を行うとき，文字変数に値を代入したり，一つの文字変数の値が次々に異なる値を取る場合があります．

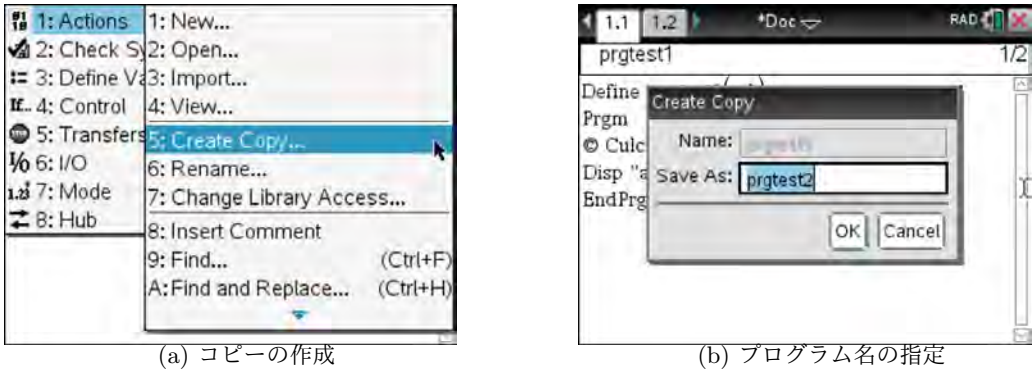
たとえば，文字変数 x に 3 という値を代入するには,「$x := 3$」とするか，または「$3 \to x$」とします．記号「$\to$」は， sto により入力することができます．x の値を 1 だけ増やすには,「$x := x + 1$」とするか，または「$x + 1 \to x$」とします．

既存プログラムのコピー

プログラムや関数を実行するとき，実行途中でユーザーによる値を求めるように指定することもできます．たとえば，プログラム prgtest1 を修正して，実行途中で b の値の入力を求めるプログラムを作ってみましょう．prgtest1 をちょっと修正するだけです．

このように，既存のプログラムや関数をちょっと修正して別なプログラムや関数を作成したいときは，既存のもののコピーを作成することができます．

最初に，プログラム画面にコピーしたいプログラムか関数を表示させます．画面が分割されているときは， doc 5 6 により不要な方を削除しておきます．prgtest1 をコピーするには，プログラム画面に prgtest1 が表示されている状態で図 41.5(a) のように menu 1 から「5: Create Copy」を選択します．(b) のようにコピーしたものの名前をどうするかが問われるので，最初に名前を入力します．デフォルトでは前のファイル名の末尾に数字が付されます．ここでは「prgtest2」とします．



(a) コピーの作成　(b) プログラム名の指定

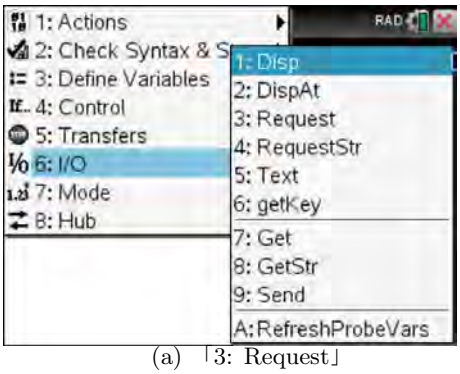
図 41.5: 既存プログラムのコピーの作成

名前を変更すると画面が左右に分割されて右側にコピー内容が表示されるので，それを修正します．左側にあるコピー元の画面を削除するには，ポインターをコピー元に移して doc 5 6 を押します．

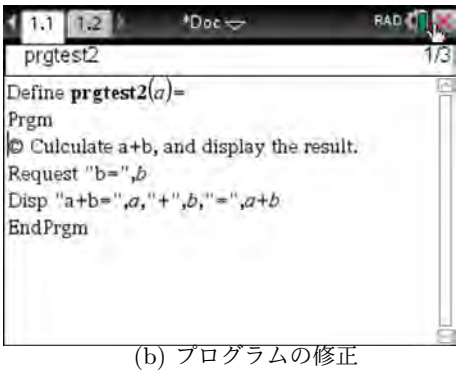
ユーザーからの数値の入力を求めるコマンドは「Request」，文字の入力は「RequestStr」です．プログラム画面の menu 6 の箇所にあります．このコマンドを利用して b の値を実行中に入力できるようにするには，たとえば「Request "b=", b」とします．「" "」で囲うことで，画面に何の入力を求めているのかを表示させることができます．

b の入力を求めるので引数は a だけです．たとえば prgtest2(3) として実行すると，図 41.6(c) のような枠が表示されて入力を求められ，「4」を入力すると (d) のような結果が表示されます．

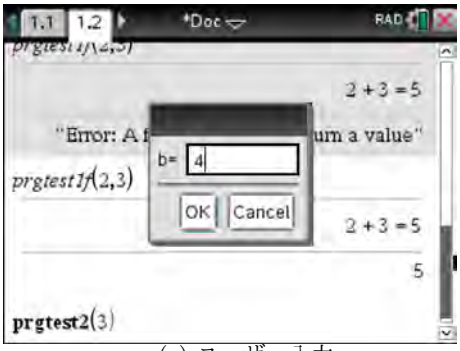
Request や RquestStr はプログラム内での利用に限られます．関数での利用はできません．



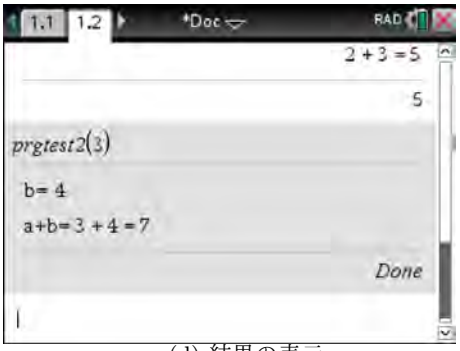
(a) 「3: Request」



(b) プログラムの修正



(c) ユーザー入力



(d) 結果の表示

図 41.6: ユーザー入力を含めての計算

なお，プログラムとして作成したものを関数に，あるいは逆に関数として作成したものをプログラムにコピーすることはできません．つまり，「Prgm ··· EndPrgm」を「Func ··· EndFunc」に，あるいはその逆の変換をすることはできません．そのような必要があるときは，内部に書かれているものを (普通に) コピーしてペーストするのがよいと思われます．コピーしたい部分を青地指定して，ctrl menu からコピーやペーストをすることができます．

または， ctrl C でコピー， ctrl V でペーストが行われます．

プログラム名の変更・プログラム内容の閲覧

一度つけた名前を変更するには，図 41.5(a) のメニューから「6: Rename」を選択することで行うことができます．「4: View」を選択すると，別なプログラムの内容を見ることができます．

ローカル変数

プログラムを作る上では，その内部でいろいろな変数を扱うことになります．内部処理のためだけに使用する変数を**ローカル変数**といいます．定義している関数やプログラムの最初で「Local $x, y, \cdots$」とすると，$x, y, \cdots$ はローカル変数として使用されます.「Local」はプログラム画面の menu 3 に登録されています.

ユーザー入力を求めるプログラム prgtest2 では，このような変数の指定は行っていません．したがって，計算画面で prgtest2 を実行後に b の値を表示させると，直前で使用した値がそのまま残っています．なお，引数として使用する変数は自動的にローカル変数として扱われます．右図のように，b の値は直前に入力した値が表示されますが，a の値は表示されません．

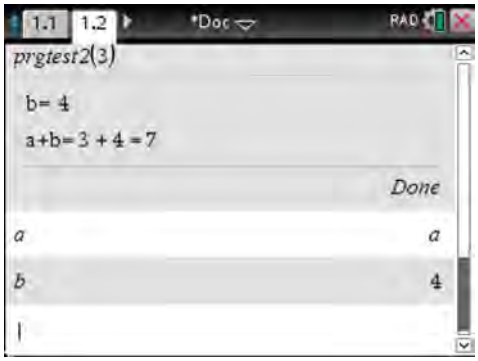


図 41.7: ローカル変数

関数やプログラムの実行のためだけに必要な変数は，プログラムの最初で「Local b」などとして，ローカル変数の宣言をしておくのがよいでしょう．

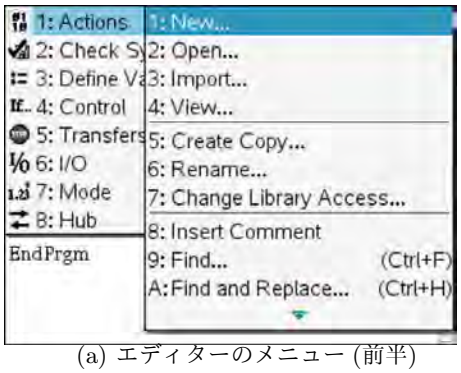
複数コマンドの実行

関数やプログラム内の記述をするときは，基本的に 1 行に 1 つの式やコマンドを書き込ます．簡単なコマンドを行を変えながら書き込んでいくと，行数が増えて管理しにくい場合もあります．

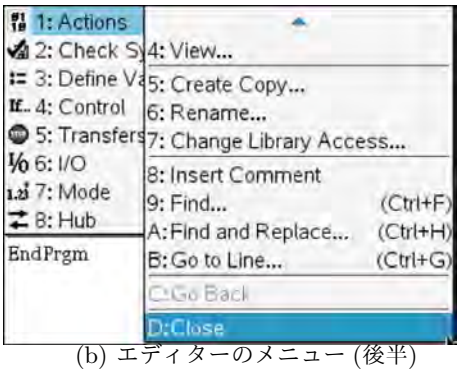
そのようなとき，記号「:」で区切りながら書き入れると，1 行に複数のコマンドを書き込むことができます．たとえば，a, b, c にそれぞれ $1, 2, 3$ を代入するとき，1 行に「$1 \to a$: $2 \to b$: $3 \to c$」または「$a := 1$: $b := 2$: $c := 3$」という書き方をすることで済ませることができます．簡単な式の計算やコマンドが連続するようなときに利用するとよいでしょう．

エディターとしての編集

プログラム画面はプログラム・エディターとしての機能があるので，通常のエディターのように利用することができます．どのようなことができるかは， menu 1 に登録されています．一つ一つ説明することはしませんが，検索 (Find)・置換 (Replace)・ジャンプ (Go to) 機能もあります．他のプログラムをインポート (import) したり，画面にオープンすることなく内容を見る (View) だけの機能もあります．いろいろと試してみてください．



(a) エディターのメニュー (前半)



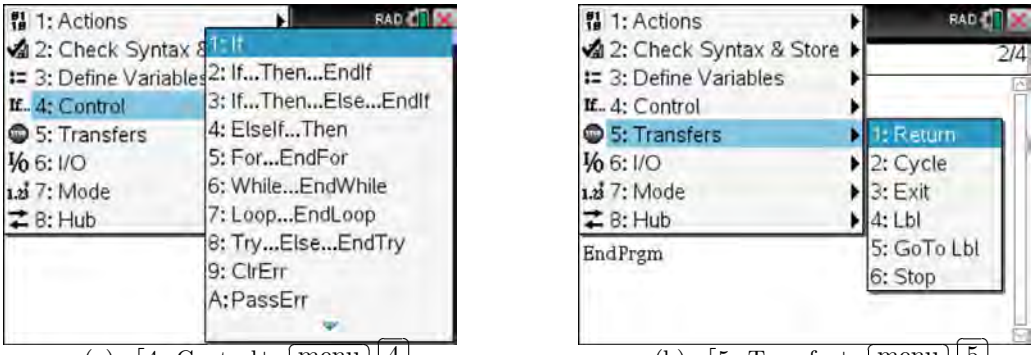
(b) エディターのメニュー (後半)

図 41.8: プログラム・エディターの機能 menu 1

41.3 いろいろな制御コマンド

この節では TI–BASIC のいろいろな制御コマンドを紹介します．ただし，その使い方の詳しい解説までは行いません．それらは通常のプログラミング言語と同様なので，使い方に関してはプログラミングに関する他の入門書を参照してください．

いろいろなコマンドは menu に登録されており，実際のプログラミングでは 4 ～ 7 をよく利用することになるでしょう．それぞれ，次のようなメニューが表示されます．



(a) 「4: Control」 menu 4　(b) 「5: Transfer」 menu 5

(c) 「6: I/O」 menu 6　(d) 「7: Mode」 menu 7

図 41.9: いろいろな制御コマンド

menu 4 には条件分岐や繰り返し処理のコマンドが，menu 5 には一つの処理から脱け出るためのコマンドが，menu 6 には画面に表示したりユーザー入力を求めるときのコマンドが，そして menu 7 には結果を表示するための書式モードに関するコマンドが登録されています．

表 41.1: menu 4　制御コマンドの使用例

制御コマンド	コメント
If (条件) 　(処理 1) (処理 2)	条件が真のときに処理 1 を実行する．処理 2 は，条件の真偽によらず実行される．条件や処理 1 が 1 行で書き切れるようなときに利用する．

If (条件) Then
　(処理 1)
EndIf
(処理 2)
	上記と同様であるが，処理 1 は複数行にわたる処理でもかまわない．条件が真のときは，処理 1 を行った後に処理 2 が実行される．偽のときは，処理 2 だけが実行される．

If (条件) Then
　(処理 1)
Else
　(処理 2)
EndIf
(処理 3)
	条件が真のときは，処理 1 の後で処理 3 を実行し，偽のときは処理 2 の後で処理 3 を実行する．処理 1 や処理 2 は複数行にわたってもかまわない．

If (条件 1) Then
　(処理 1)
ElseIf (条件 2)
　(処理 2)
EndIf
(処理 3)
	条件 1 が真のときは，処理 1 の後で処理 3 を実行する．偽のとき，さらに条件 2 が真のときは処理 2 の後で処理 3 を実行する．処理 1 や処理 2 は複数行にわたってもかまわない．
	つまり，処理 3 は必ず実行される．処理 1 は条件 1 が真の場合に実行され，処理 2 は条件 1 が偽で条件 2 が真のときに実行される．

For $v1, v2, v3, v4$ 　(処理 1) EndFor (処理 2)	変数 $v1$ の値を，$v2$ の値から $v3$ の値まで $v4$ ずつ増やしながら処理 1 を実行する．$v1$ の値が $v3$ の値を超えたら，処理 1 の実行を中止して処理 2 を実行する．$v4$ を省略すると $v4 = 1$ として実行される．変数 $v1$ は，事後に必要がなければローカル変数として宣言する．$v4 < 0$ とすることもでき，そのときは $v2 > v3$ に設定する．

(値 $\to v$)
While (条件)
　(処理 1)
EndWhile
(処理 2)
	条件が真である間、処理 1 の実行を繰り返す．条件が偽になったら，処理 1 の実行を中止して処理 2 を実行する．条件の真偽を判断するための変数の値を最初に設定し，処理 1 の中ではその変数の値を変更する処理を含む必要がある．

Loop 　(処理 1) EndLoop (処理 2)	処理 1 を繰り返す．処理 1 は，その処理から抜け出るための何らかのコマンド (If, Exit, Go, Label 等) を含む必要がある．処理 1 の繰り返しを終えた後で処理 2 が実行される．

制御コマンド	コメント
Try 　(処理 1) Else 　(処理 2) EndTry	処理 1 がエラーを含まなければそれを実行する．処理 1 がエラーを含むときは処理 2 を実行する．エラーが生じたとき，そのエラーコードが errCode に記録される．その内容は「Refference Guide」を参照することで知ることができる．
ClrErr	エラーの状態をクリアして，errCode の値を 0 にする．
PassErr	エラーを，Try ··· EndTry 内の次のブロックに引き渡す．

表 41.2: menu 5 移動コマンドの使用例

移動コマンド	コメント
Return	「Func ··· EndFunc」内で利用することにより，その関数の値を返す.「Prgm ··· EndPrgm」内では利用できない．
Cycle	このコマンドが For，While，Loop などの繰り返し処理の中にあると，Cycle 以降の処理を省略して次の繰り返しに戻る．繰り返し処理以外の箇所では使用できない．
Exit	このコマンドが For，While，Loop などの繰り返し処理の中にあると，繰り返し処理を終了して，繰り返し後の処理に移る．
Lbl (ラベル名) (処理) Goto (ラベル名)	プログラムのある行にラベルを付することで，別な箇所からラベルを付した行の処理に飛ぶことができる．
Stop	このコマンドがプログラムの中にあると，その箇所でプログラムの実行を強制終了する．関数の中では利用できない．

「Cycle」の使用例として，Reference Guide では右図のプログラムが紹介されています．

Cycle を含まなければ，単に temp の値を 1 から 100 まで 1 ずつ増やしながら和を計算しているだけなので,「Disp temp」で表示される値は $1+2+\cdots+100=5050$ となります．しかし，Cycle があると，$i=50$ のときに Cycle の下の和の計算を行わないで次の繰り返しに移るので,「Disp temp」で表示される値は 5050 より 50 だけ少ない「5000」が表示されます．

```
0 → temp
For i=1, 100, 1
If i=50
Cycle
temp+1→ temp
EndFor
Disp temp
```

表 41.3: menu 6 入出力コマンドの使用例

入出力コマンド	コメント
Disp ”文字列”, x	Disp の後に文字列や数値を表示する．文字列は” ”で囲う．文字変数が値を持つときは，その値が表示される.「,」で区切ることで続けて表示することができる．
Request ”文字列”, 数値	枠を開いて，指定した文字列を表示してユーザーからの数値の入力を求める．
RequestStr ”文字列”, 文字	枠を開いて、指定した文字列を表示してユーザーからの文字の入力を求める．
Text (文字列)	プログラム実行中に，枠を開いて，プログラム内で事前に作成した文字列を表示する．数値は，いったん文字に変換してから表示させる．枠は,「Ok」が押されるまで表示される．

図 41.9(c) では，他に「Get」「Send」などのコマンドがあります．これらは Vernier DataQuest を利用しているときのコマンドで，センサーとの信号のやりとりに関するコマンドです．

図 41.9(d) は，表示する小数の桁数や角の単位などを変更するときに利用します．この箇所は，表示されるサブメニューをみれば，どのような内容であるかは明らかと思われます．

表 41.4: menu 7 モード変更コマンド

モード変更コマンド	コメント
Display Digits	小数を表示するとき，その桁数を指定する．
Angle	角の単位を，Radian, Degree, Gradian から指定する．
Exponential Format	指数表記の仕方を，Normal, Scientific, Engineering から指定する．
Real or Complex	複素数の表記の仕方を，Real, Rectangular, Polar から指定する．
Auto or Approx	計算の仕方を，Auto, Approximate, Exact から指定する．
Vector Format	ベクトルの表記を，Rectangular, Cylindrical, Spherical から指定する．
Base	基数を，Decimal, Hex, Binary から指定する．
Unit System	単位を扱うとき，単位を SI, Eng/US から指定する．

41.4　文字列の処理

文字列・リストの処理に関しては，次のようなコマンドが用意されています．これらのコマンドは計算画面やテーブル画面などでも利用することができます．

以下の表では文字列に関して示しますが，$\{a, b, c, \ldots\}$ のようなリストでも同様の処理が行われます．

表 41.5: 文字列・リストの処理に関するコマンド

文字列処理のコマンド	コメント
&	文字列を「&」で繋いで複数の文字列を接続し，一つの文字列に変換する．ex) ”ab” & ”cd” ⇒ ”abcd”
#	文字列を変数に変換する．ex) #(”var”) ⇒ var
char(文字コード)	文字コードに対応する文字を返す．ex) char(65) ⇒ ”A”
dim(文字列)	文字列の個数を返す．ex) dim(”abcd”) ⇒ 4
expr(文字列)	文字列を数式に変換する．文字列の中に expand 等の計算コマンドが含まれると，変換後のコマンドが実行される．ex) expr(”expand((x+1)^2”) ⇒ $x^2 + 2 \cdot x + 1$
string(数式)	数式を文字列に変換する．ex1) string(1.234) ⇒ ”1.234” ex2) string($\sqrt{x} \cdot sin(x)$) ⇒ ”√ (x)·sin(x)”
format(数値, 書式)	数値を指定された書式の数値文字列に変換する．書式は，”fn”, ”sn”, ”en”などで指定する．n には具体的な数を入れる．”f”は固定小数点，”s”は科学的表記，”e”は工学的表記である．ex) format(123.4567, ”s3”) ⇒ ”$1.235_{\mathrm{E}}0$”
inString(文字列 1, 文字列 2)	文字列 1 の中に文字列 2 が含まれるかどうかを調べ，含まれるときは左から何番目にあるかという数を返す．含まれないときは 0 を返す．ex) inString(”abcd efg”, ”fg”) ⇒ 7
Left(文字列, n)	文字列の，左から n 番目までの文字を返す．ex) Left(”abcde”, 3) ⇒ ”abc”
Right(文字列, n)	文字列の，右から n 個の文字を返す．ex) Left(”abcde”, 3) ⇒ ”cde”
mid(文字列, n_1, n_2)	文字列の，左から n_1 番目の文字から数えて n_2 個の文字を返す．n_2 を省略したときは，n_1 番目以降の文字を全部返す．
ord(文字列)	文字列の，最初の文字の文字コードを返す．ex) ord(”ABCD”) ⇒ 65
rotate(文字列,n)	文字列の文字をずらす．$n > 0$ のときは左側に n 個ずれて，あふれた文字は右側に移る．$n < 0$ のときは右側にずれて，あふれた文字は左側に移る．省略すると $n = -1$ が仮定される．
shift(文字列, n)	n が省略されると，右側に 1 つシフトして右端 1 文字が欠ける．$n > 0$ のときは左側に n 回シフトして，左側 3 文字が欠けて右端に空白 3 文字が入る．$n < 0$ のときは右側が欠ける．

(注) rotate と shift は，2 進数や 16 進数で表された数に関しても有効です．

41.5　サブルーチン

プログラムの中では，Nspire の持ついろいろなコマンドに加えて，LibPrib や LibPub を指定して自分で定義したプログラムも利用することができます．また，プログラムの中に別なプログラムを含めることもできます．そのような内部プログラムは**サブルーチン**と呼ばれます．

サブルーチンは，それが実行される前に定義されている必要があります．たとえば，右図のような形で定義します．下記のことに留意してください．

- サブルーチンの定義は，本処理の前に定義する．
- 複数のサブルーチンを定義してもかまわない．
- サブルーチンの名前は，ローカル変数として宣言する．
- サブルーチン内では，本処理で使用するローカル変数を利用することはできない．
- 本処理では，サブルーチン内で使用するローカル変数を使用することはできない．

```
Define sample()=
Prgm
local subsample
Define subsample()=
Prgm
      (内部処理)
EndPrgm
(本処理)
EndPrgm
```

41.6　プログラム例

方程式 $f(x)=0$ の実数解

方程式 $f(x)=0$ が区間 $[a,b]$ の中に 1 個の実数解を持つ場合に, その実数解を求めるプログラムを作ってみましょう．ただし，関数 $f(x)$ はすでに定義済みで $f(a)f(b)<0$ とし，その中の実数解は 1 つだけであるものとします．両端の関数の値が異符号の状態を保ったまま，小区間の幅を狭めていくことで近似解を求めます．関数として作成すると，たとえば次のようになります．

プログラム	解説
Define kai(a,b)=	プログラム名を kai とする．引数は, 区間 $[a,b]$
Func	関数であることを宣言
Local aa, bb, ee, mm, yy	ローカル変数の宣言
a→aa	a の値を aa に代入する
b→bb	b の値を bb に代入する
(a+b)/2→mm	$(a+b)/2$ の値を mm に代入する
10 ^ (−6)→ee	誤差の限界を 10^{-6} にして ee に代入する
While abs(f(mm))>ee	$\lvert f(mm)\rvert > ee$ ならば以下を実行する
If f(aa)∗f(mm)>0 Then	$f(aa)$ と $f(bb)$ が同符号ならば, 以下を実行する
mm→aa	mm の値を aa に代入する
Else	$f(aa), f(bb)$ が同符号でないときは 以下を実行
mm→bb	mm の値を bb に代入する
EndIf	If 文終わり
(aa+bb)/2→mm	$(aa+bb)/2$ を mm に代入する
EndWhile	While 文終わり
Return approx(mm)	誤差 10^{-6} 以内で求めた実数解 mm を返す
EndFunc	関数定義の終了

(注) 「approx(x)」は，x の近似値を表示させるコマンドである．

シンプソンの公式

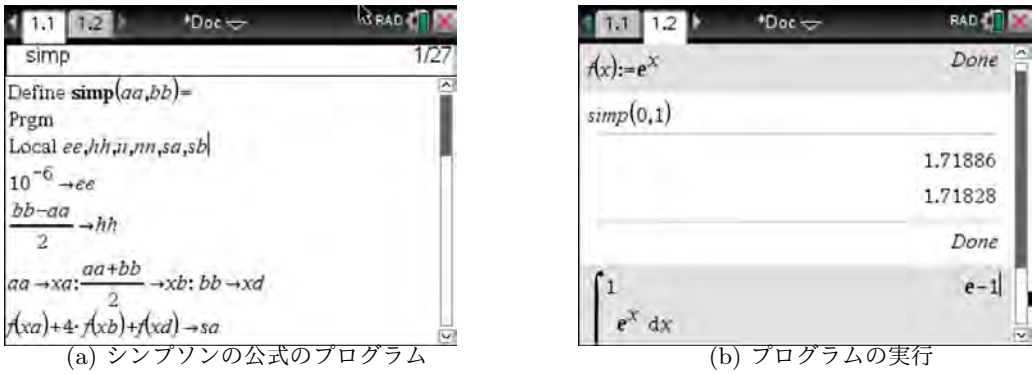
プログラム例として, 数値積分の場合も紹介しておきます。シンプソンの公式は, 区間 $[a,b]$ を $2n$ 等分して分点を (x_k,y_k) $(0 \leqq k \leqq 2n)$, 区間の幅を $h=(b-a)/2n$ とするとき, 定積分の値を

$$\int_a^b f(x)dx \fallingdotseq \frac{h}{3}\{y_0+4(y_1+y_3+\cdots+y_{2n-1})+2(y_2+y_4+\cdots+y_{2n-2})+y_{2n}\}$$

により近似計算するものです。右辺は, $\displaystyle\sum_{k=0}^{n-1}\frac{h}{3}(y_{2k}+4y_{2k+1}+y_{2k+2})$ を計算することにより得られるので, たとえば以下のようなプログラムになります。ただし, $f(x)$ はすでに定義済みとします。

プログラム	コメント
Define simp(aa,bb)=	プログラム名を simp とし，区間 $[aa,bb]$ で考える
Prgm	プログラムであることを宣言
local ee, hh, ii, nn, sa, sb	ローカル変数であることの宣言
local xa, xb, xd, ya, yb, yd	xc, yc はシステム変数で利用されるので省く
10 ^ (−6)→ee	誤差の限界を 10^{-6} にし, ee に代入
(bb−aa)/2→hh	区間の中点の座標を hh に代入
aa→xa : (aa+bb)/2→xb : bb→xd	区間を 2 等分したときの座標を求める
f(xa)+4f(xb)+f(xd)→sa	sa を計算
hh∗sa/3→sa	sa は区間を 2 等分した場合の初期値
Disp approx(sa)	面積の初期値 sa を小数表示する
2→nn	$nn=2$ として 4 等分から繰り返し計算を開始
Loop	以下の内容を繰り返す
0→sb : aa→xa	初期値の設定
(bb−aa)/(2∗nn)→hh	区間の幅の計算
For ii, 1, nn, 1	繰り返し変数を ii とし, 以下を nn 回繰り返す
f(xa)→ya	$ya=f(xa)$ とする
f(xa+hh)→yb	$yb=f(xa+hh)$ とする
f(xa+2∗hh)→yd	$yd=f(xa+2hh)$ とする
sb+(ya+4∗yb+yd)/3→sb	$(ya+4yb+yd)/3$ を sb に加える
xa+2∗hh→xa	計算する小区間の左端を次の計算用に変更する
EndFor	ii による繰り返しの終了
hh∗sb→sb	sb が, $2nn$ 等分した場合の面積
If abs(sa−sb)<ee Then	sa との差が ee 以下なら以下を実行する
Exit	Loop から脱出する
Else	$\lvert sa-sb\rvert \geqq ee$ のときは以下を実行
ⓒDisp approx(sb)	途中経過を知りたいときはⓒ を削除する
sb→sa : nn+1→nn	sb を直前の値 sa とし, 分割数を増やす
EndIf	If 文を終了する
EndLoop	Loop 文を終了する
Disp approx(sb)	最終結果を表示する
EndPrgm	プログラムの終了

図 41.10(a) は，プログラム画面の最初の部分です．(b) は，$f(x) = e^x$ を最初に定義して，simp(0,1) により $\int_0^1 f(x)\,dx$ の近似値を求めています．最後の行では，Nspire のコマンド利用してこの定積分の値を求めています．$e - 1 = 2.71828459\cdots - 1 = 1.71828\cdots$ となります．



(a) シンプソンの公式のプログラム　　(b) プログラムの実行

図 41.10: シンプソンの公式による定積分の計算

これらの例をみると，通常のプログラムと同様であることが分かるでしょう．プログラムの作成練習や計算のアルゴリズムの理解のために，このようなプログラムを作成してみることは有用です．しかし，単に定積分 $\int_a^b f(x)dx$ の値を求めたいのであれば, 計算画面で $\int(f(x), x, a, b)$ というコマンドを打ち込むだけで済みます。TI–BASIC のプログラムでは, このような数式処理のコマンドもプログラムの中で利用できるので, 作成できるプログラムの範囲が大きく広がることになります。その意味で, 計算部分に関しては Nspire の TI–BASIC は C や BASIC など他の言語より強力なプログラム言語といえるでしょう．

スクリプト言語「Lua」

この章の冒頭でも述べたように，Nspire の TI–BASIC にグラフ描画関係のコマンドはありません．それらは，Student Software のアプリケーションである「Script Editor」を利用して，TI–BASIC とは異なるプログラミング言語である「Lua」を利用して作成することになります．Lua にはグラフ描画関係のコマンドが備わっています．

「Lua」については下記を参照してください．実際に試してみるには，パソコンに「Lua」をインストールして行った方が良いです．Script Editor には，基本的な io.read() すら備わっていません．

[1] Lua　(Lua 本体をダウンロードできます．フリーソフトです．)
　　http://www.lua.org/
[2] 高速スクリプト言語「Lua」を始めよう！　(インストールの仕方から解説されています．)
　　http://www.hakkaku.net/series/
[3] Lua プログラミング入門　(基礎から解説されています．)
　　https://densan-labs.net/_downloads/lua.pdf
[4] お気楽 Lua プログラミング超入門　(他の言語と比較しながら解説されています．)
　　http://www.geocities.jp/m_hiroi/light/lua.html