

HP Prime Programming Tutorial

Edward Shore

Hardware Rev. 13441

All original content copyright, © 2013, 2018. Edward Shore. Unauthorized use and/or unauthorized distribution for commercial purposes without express and written permission from the author is strictly prohibited. This tutorial may be distributed for noncommercial purposes, provided that full credit is given to the author. This document is provided "as-is". Please contact the author at ews31415@gmail.com if you have questions.

Calculator screen shots are generated with the HP Prime Virtual Calculator, © 2018 HP Development Company, L.P. The author is not an employee of Hewlett Packard.

Rev. 2

HP Prime Programming Tutorial

Table of Contents

Introduction.....	4
Tutorial #1: LOCAL, RETURN.....	5
The RETURN Command.....	6
Checking Programs.....	6
Running Programs.....	6
Storing Values to Variables.....	7
The LOCAL Command.....	7
The Program MOPMT.....	8
Tutorial #2: MSGBOX, IF-THEN-ELSE, PRINT, when, FOR.....	9
The Command MSGBOX	9
The Program COMLOCK.....	9
The IF-THEN-ELSE Program Structure.....	9
The PRINT Command	10
The when Command.....	11
The Program DOESTAX.....	11
The Basic FOR Loop.....	11
The Program DISPCUBES.....	12
The Program SUMDIV.....	12
The FOR Loop with STEP.....	13
The Program PRINTEVENS.....	13
Tutorial #3: WHILE, INPUT, KILL, REPEAT, GETKEY.....	15
The WHILE Structure.....	15
The INPUT Command (Basic).....	15
The Program TARGET.....	16
INPUT with a List of Choices.....	16
The Program EARTHCOORD.....	16
INPUT with Checkboxes.....	17
The Program DOESTAX2.....	17
The REPEAT Structure.....	18
The Program ULAM.....	18

The GETKEY Command.....	19
The Program KEYNO.....	20
Tutorial #4: CHOOSE and CASE.....	21
The CHOOSE Command.....	21
The Program TERMVEL.....	21
The CASE Structure.....	22
The Program AREAC.....	22
Tutorial #5: STARTAPP, STARTVIEW, CHECK, UNCHECK, RGB, HAngle.....	24
System Equation Variables.....	24
Defining Equations from Home and Inside Programs.....	24
The STARTAPP Command.....	24
The CHECK and UNCHECK Commands.....	25
The STARTVIEW Command.....	25
Designating Colors with RGB and Hex Codes.....	26
The Color of a Graph.....	27
The System Variable HAngle.....	27
The Program CONIC.....	27
The Program PROJ13.....	28
Tutorial #6: Subroutines.....	30
How Subroutines Work.....	30
The Program SUBEXAM.....	30
Tutorial #7: Graphics Screen, TEXTOUT.....	32
The Graphics Screen: Cartesian vs Pixel.....	32
The Commands TEXTOUT and TEXTOUT_P.....	35
The Program SNOWFLAKE.....	36
Tutorial #8: LINE, FILLPOLY, ARC.....	38
The LINE Command.....	38
The Program DRAWHOUSE.....	38
Drawing Polygons with the FILLPOLY Command.....	39
The Program PENT.....	39
The ARC Command.....	40
Drawing Circles with ARC.....	40
Drawing Ellipses with ARC.....	40
Drawing Partial Arcs with ARC.....	41
The Program DRAWARCS.....	41

Tutorial #9: Numerical Calculus with the Template Key.....	43
Numerical Derivatives.....	43
Definite Integrals.....	43
Double Definite Integrals.....	43
Summation.....	43
Substitution.....	43
The Program CALCDEMO.....	44

Introduction

This is an in-depth tutorial of the HP Prime HP PPL, which is the primary programming language for the HP Prime. In late 2013, I posted a series of tutorials on my blog, Eddie's Math and Calculator Blog, <https://edspi31415.blogspot.com/>, which covered many basic programming topics.

Most of the text will be coming from these blog entries, with some additional information added to this document.

I hope you enjoy this tutorial and allows you to get a firm foundation of the programming language of the HP Prime.

For this tutorial, the current firmware, 13441 (2/12/2018) will be used.

Key Conventions:

Physical Keys: [enclosed in brackets]

Soft Touch Menu Keys: (enclosed in parenthesis)

Happy Programming,

Eddie Shore

Tutorial #1: LOCAL, RETURN

Let's get started! By the end of this tutorial session, you will be making programs and learning how to run them.

How To Start Writing a HP Prime Program

How to start writing a program:

1. Press the [Shift] key, then the [1] key.
2. You will see a list of all the programs that have already been entered in memory. To start a new program, press the soft (New) key.
3. Enter the name of the program. To enter a letter, press the [ALPHA] key and corresponding letter that goes with the key. Pressing the [ALPHA] key twice will turn on UPPERCASE ALPHA-LOCK. If you want to type lower case letters, press [ALPHA], [Shift], [ALPHA] to turn on lowercase alpha-lock. To exit any lock, press the ALPHA key one more time. When happy with the name, press [Enter].

Rules for Program Names:

1. The only characters allowed in a program name are: letters, Greek letters, numbers, and the underscore character (_).
2. The program name must start with a letter. For example: TEST123 is a valid name, TEST_123 is also a valid name, but neither 123TEST or 123_TEST are not valid program names.

Structure of a HP Prime Program

A HP PPL program is encased of an EXPORT - BEGIN - END structure. The layout is generally like this:

```
EXPORT program_name(arguments)  
BEGIN  
  commands and comments go here  
END;
```

Each line containing a command generally must end with a semicolon (;). A semicolon can be typed by pressing [ALPHA], [+].

Comments are statements included in the program to give the user and the programmer useful notes and reminders while looking over the program. Comments can also divide and name sections. To type a comment, you will need two forward slashes (//), which can be generated by pressing the divide key [÷]. Anything in the line following the two slashes is ignored in running the program.

Now let's look at our first program, SQIN. This program will first square a number, then take the reciprocal of it.

The RETURN Command

The Program SQIN

SQIN takes a number, squares it, then calculates the reciprocal. In short we are defining a custom function:

$$\text{SQIN}(x) = 1/x^2$$

Commands:

RETURN: returns a result to the stack (home page). You can return numbers, lists, vectors, matrices, strings, or a combination of these times.

Access: (Tmpl), 1. Block, 2. RETURN

All the program code in this tutorial series will be shown in Courier font.

```
EXPORT SQIN(X)
BEGIN
RETURN 1/X^2;
END;
```

Checking Programs

You can check the syntax of the program just by pressing the Check soft key in the program editor. HP Prime will inform you if there is a syntax error and attempt to point you to the error. If there are no syntax errors, the Prime states "No errors in the program". I use the Check command all the time.

Caution: checking programs do not check whether the arguments used by the commands are valid. However, it is still a good way to check your progress.

Running Programs

Home Mode - Textbook Entry,
Home Mode - Algebraic Entry,
CAS Mode:

Type the program name, follow the name with parenthesis and enclose the required arguments.

Or use the Toolbox (top row of white keys, 2nd key from the left, it looks like a tool box, in this tutorial I will type the key as [Toolbox]), select the User touch key, select the program, and input

the required arguments.

Home Mode - RPN Entry:

Enter each argument, separate each entry by pressing the Enter key. Type the name, and in the parenthesis state the number of arguments.

For example, if the program TEST has four arguments, the RPN stack would like this:

```
4: argument_1
3: argument_2
2: argument_3
1: argument_4
TEST(4) to run the program.
```

Examples to try with SQIN:

```
SQIN(5) returns .04
SQIN(36) returns .000771604938
```

Storing Values to Variables

There are two ways to store values to variables. First is to use the [[Shift](#)], [EEX] (Sto>) method:

```
4 > A
```

The other is to use the equals then colon method (:=), like so:

```
A:=4
```

The programs listed in this tutorial will use the latter method.

The next program will demonstrate the concept of local variables.

The LOCAL Command

Local variables are variables that are belong to the program or subroutine. When the program ends, local variables are deleted.

There are two methods of declaring local variables: using variables as pass-through arguments, or using the LOCAL command. Variables using the LOCAL command can be any type: real numbers, complex numbers, strings, lists, matrices, integers, etc.

Tip: When variables are declared using LOCAL, the default value given to these variables is 0.

The other type of variable are global variables, which can be defined inside or outside a program and retain their value. Global variables include system variables, and some global variables have restrictions.

Here we will focus on local variables.

Access: (Tmpl), 4. Variable, 1. LOCAL

Tip: You can declare local variables and assign an initial value at the same time. For example: LOCAL K:=1; stores 1 in K and makes K a local variable.

The Program MOPMT

MOPMT calculates the monthly payment of a loan. The arguments are: the loan amount (L), the interest rate (R), and the number of months (M).

```
EXPORT MOPMT (L,R,M)
BEGIN
LOCAL K:=R/1200;
K:=L*K/(1-(1+K)^-M);
RETURN "Payment "+K;
END;
```

Since L, R, and M are pass-through arguments, they are automatically local variables. K is declared a local variable. When MOPMT finishes, nothing globally is stored to L, R, M, or K.

Tip: Use RETURN, TEXTOUT_P, and PRINT to return custom strings, which combine results, messages, and calculations. Strings are connected with a plus sign.

Examples:

MOPMT(4000, 9.5, 30) returns 150.317437565

MOPMT(370000, 3.5, 360) returns 1661.46534383

Tutorial #2: MSGBOX, IF-THEN-ELSE, PRINT, when, FOR

The Command MSGBOX

MSGBOX takes a string and makes a pop-up message box. Program execution stops until you press a key to acknowledge the message. The pop-up message is a singular string.

Access: (Cmds), 6. I/O, 8. MSGBOX

The Program COMLOCK

Imagine that you are in charge of setting the combinations for the good, old-school combination locks. This program gives three digit combinations of integers 0 through 39, and uses MSGBOX to return the combination.

```
EXPORT COMLOCK ()
BEGIN
LOCAL L0;
L0:=RANDINT(3,0,39);
MSGBOX("SECRET: "+L0(1)+"", "+L0(2)+"", "+L0(3)");
END;
```

Featured command: RANDINT(n, a, b) generates a list of n integers between a and b. You can leave n out if you desire a single random integer. Picks may be repeated.

Access: [Toolbox], (Math), 5. Probability, 4. Random, 2. Integer

The IF-THEN-ELSE Program Structure

IF-THEN-ELSE: Program structure:

```
IF condition THEN
  do if the condition is true;
ELSE
  do if the condition is false;
END;
```

Access: (Tmpl), 2. Branch, 2. IF THEN ELSE

Tip: You can leave out the ELSE part if you only want to test to see if a condition is true. Access the simple IF-THEN structure by pressing (Tmpl), 2. Branch, 1. IF THEN.

Access the testing symbols $<$, \leq , $=$, \neq , \geq , and $>$; along with AND, OR, by pressing [Shift], [6].
Note that the double equals is needed to check equality.

Comparison tests result in either 1 for true, or 0 for false. AND, along with OR, can be used to combine to conditions. For example, if you want to test whether X is in the range of 1000 and 1999 inclusive, use the testing condition $(X \leq 1000) \text{ AND } (X \leq 1999)$.

The PRINT Command

The PRINT command prints a string, result, or a combination of both onto the Prime's Terminal screen. If PRINT is used, the program will end on the terminal (text output) screen. Press any button to exit.

Access: (Cmds), 6. I/O, 9. PRINT

To clear the terminal screen, type PRINT(). This is a good way to clear the terminal screen and I usually use this at the beginning of any program if PRINT is going to be used later on.

You can access the terminal screen at any time by pressing and holding the [ON], then press [÷].

The Program QROOTS

The program QROOTS (yet one more quadratic solver, sorry for not being original guys and gals), demonstrates the use of IF-THEN-ELSE and PRINT.

Here I set the setting variable HComplex to 1, which allows for complex number results.

```
EXPORT QROOTS (A,B,C)
BEGIN
LOCAL D;
PRINT();
HComplex:=1;
D:=B^2-4*A*C;
IF D≥0 THEN
PRINT("Roots are real.");
ELSE
PRINT("Roots are complex.");
END;
PRINT((-B+√D)/(2*A));
PRINT((-B-√D)/(2*A));
END;
```

Examples:

QROOTS(1,5,8) returns:

Roots are complex.

-2.5+1.32287565553*i

-2.5-1.32287565553*i

QROOTS(2,-4,-8) returns:

Roots are real.

3.2360679775

-1.2360679775

The when Command

When the instructions for *both* the THEN and ELSE is one command, we can use the shortcut command `when` (note the lowercase spelling). The command is treated as a function. Access the when command in the general catalog.

Syntax:

```
when(condition, do this command if true, do this command if false)
```

The Program DOESTAX

The DOESTAX program tests whether sales tax applies to an item given its price. Assume that there is an 8.5% sales tax when the item's price exceeds \$10.00, otherwise no sales tax applies.

```
EXPORT DOESTAX(N)
BEGIN
N:=when(N>10,N*1.085,N);
RETURN N;
END;
```

Examples:

DOESTAX(8) returns 8

DOESTAX(12) returns 13.02

The Basic FOR Loop

This section will explore the basic **FOR** structure:

```
FOR variable FROM start TO end DO
commands;
END;
```

All the commands in the loop will be executed a set number of times. Each time a loop finishes, the *variable* increases by one. The loop terminates when *variable=end*.

Access: (Tmpl), 3. LOOP, 1. FOR

The Program DISPCUBES

The program DISPCUBES prints out all the cubes (x^3) for integers 1 to N.

```
EXPORT DISPCUBES(N)
BEGIN
LOCAL I;
// clear terminal
PRINT();
// basic FOR loop
FOR I FROM 1 TO N DO
PRINT(I^3);
END;
END;
```

Example: DISPCUBES(5) returns on the terminal:

```
1
8
27
64
125
```

The Program SUMDIV

The program SUMDIV takes any integer and adds up the sum of its divisors. For example, the divisors of 12 are 1, 12, 2, 3, 4, and 6. The sum is 28.

```
EXPORT SUMDIV(N)
BEGIN
LOCAL S:=0,K,mdiv,ldiv;
mdiv:=CAS.idivis(N);
ldiv:=DIM(mdiv);
FOR K FROM 1 TO ldiv(1) DO
S:=S+mdiv(K);
END;
RETURN S;
END;
```

Examples:

SUMDIV(12) returns 28.

SUMDIV(24) returns 60.

SUMDIV(85) returns 108.

Featured Commands in SUMDIV:

idivis: `idivis(integer)` returns a sequence of all of the divisors of *integer*.

Access: [Toolbox], (CAS), 5. Integer, 1. Divisors

Any CAS command used in programming will be preceded by "CAS."

DIM: returns the dimensions of a sequence, string, or matrix. Note that DIM does not cover lists. To find the dimension of a list, use SIZE instead.

For sequences or vectors, DIM returns the length in a list {length}.

For strings, DIM returns length as a number.

For matrices, DIM returns the list {number of rows, number of columns}.

Access: (Cmds), 1. Strings, 9. DIM

The FOR Loop with STEP

The basic FOR loop has increments of 1. You can make the increment (or decrement) any amount you want.

```
FOR variable FROM start TO end STEP increment DO
  commands;
END;
```

All the commands in the loop will be executed a set number of times. Each time a loop finishes, the *variable* increases by one. The loop terminates when *variable* \geq *end*.

Access: (Tmpl), 3. LOOP, 2. FOR STEP

If you want any FOR loops with decrements, use the FOR DOWN(TO) and FOR DOWN(TO) STEP structures.

The Program PRINTEVENS

The program PRINTEVENS will print all even integers between A and B. A and B can have non-integer input. In this program, a FOR loop with increment 2 (STEP 2) is used.

```
EXPORT PRINTEVENS(A,B)
BEGIN
// Print all even numbers between
// A and B
LOCAL I;
// Clear terminal screen
PRINT();
// Check if A is even
// Take A to the next integer
```

```
A:=CEILING(A);  
// Test to see if A is even  
A:=when(FP(A/2)==0,A,A+1);  
// For loop with step increment 2  
FOR I FROM A TO B STEP 2 DO  
PRINT(I);  
END;  
END;
```

Example:

PRINTEVENS(3,10)

Result:

4
6
8
10

Tutorial #3: WHILE, INPUT, KILL, REPEAT, GETKEY

I hope you are ready for the intensity. :)

The WHILE Structure

The WHILE loop repeats a set of instructions while a condition is met. Once it isn't met, the loop is.

```
WHILE condition is true DO
commands
END;
```

Access: (Tmpl), 3. Loop, 5. WHILE

Caution: Watch your ENDS! Make sure an END is with each loop and the program itself. Press the soft key Check to check your work.

The INPUT Command (Basic)

INPUT: Creates an input screen for variables. On the HP Prime, the input can be asked for more than one input. TARGET demonstrates INPUT with one prompt.

One Variable:

```
INPUT(variable, "title", "label", "help text", reset value, initial
value)
```

Multi-Variable:

```
INPUT(list of variables, "title", list of "labels", list of "help
text", list of reset values, list of initial values)
```

Note: Everything after the variable is optional. However, the order of arguments must be maintained. For example, if you want a list of initial values but no reset values, leave a blank list { } for reset values.

Note: Pressing Cancel will store a 0 in variable. You may include code of what to do if the user presses Cancel, but it is not required.

Access: (Cmds), 6. I/O, B. INPUT

There are a lot of things we can do the INPUT command.

The KILL Command

KILL: Terminates program execution. Nothing dies, I promise.

Access: (Tmpl), 1. Block, 3. KILL

The Program TARGET

HP Prime Program: TARGET. TARGET is a game where you provide a guess to get a desired number. If you miss, the calculator will tell you if number is higher and lower. At the end of the game, the calculator gives you how many picks you needed to get the target number.

```
EXPORT TARGET()
BEGIN
LOCAL C:=0, N:=RANDINT(1,20), G:=-1;
WHILE G≠N DO
C:=C+1;
INPUT(G,"Guess?","GUESS:","1 - 20");
// Guess of 0 terminates program
IF G==0 THEN
KILL;
END;
// Guess is lower than N
IF G < N THEN
MSGBOX("Higher");
END;
// Guess is higher than N
IF G > N THEN
MSGBOX("Lower");
END;

END;
// Display player's score
MSGBOX("Correct! Score: "+C);

END;
```

INPUT with a List of Choices

We can include a series of choices in INPUT box. The variable represents the slot where the choice belongs. For example, variable = 1 if the first choice is selected, variable = 2 if the second is selected, and so on. For one variable, the general syntax is:

One Variable:

```
INPUT({{variable, list of choices}}, "title", "label", "help text",
reset value, initial value)
```

Note will need two sets of brackets! Also keep in mind, everything after the variable is optional.

The Program EARTHCOORD

The program EARTHCOORD gives the user a choice of four cities. Once that is selected, you get the approximate latitude and longitude.

```
EXPORT EARTHCOORD()
BEGIN
// Localize variables
LOCAL L1,L2,L3,I;

// Store latitudes
L1:={34.2166666667,41.9,-33.8599722222,
35.6895055556};

// Store Longitudes
L2:={-118.25,12.5,151.211111111,
139.708366667};

// Store list of locations
L3:={"Los Angeles","Rome","Sydney",
"Tokyo"};

// Input
INPUT({{I,L3}}, "Lat/Long", "City:");

// Display coordinates
RETURN {→HMS(L1(I)),→HMS(L2(I))};

END;
```

INPUT with Checkboxes

We can include make a checkbox with the INPUT command. The variable associated with the checkbox returns a value of either 0 (unchecked) or 1 (checked). For one variable, the basic general syntax is:

One Variable:

```
INPUT({{variable, 1}}, "title", "label", "help text", reset value,
initial value)
```

Note will need two sets of brackets! And remember, in an INPUT command, everything after the variable is optional.

The Program DOESTAX2

Remember DOESTAX in Tutorial 2? In this version, DOESTAX2 will analyze up to three items and ask for the sales tax rate. The checkboxes below will indicate whether the tax applies.

```
EXPORT DOESTAX2()
BEGIN

// Localize variables
```

```

LOCAL item1, item2, item3;
LOCAL ck1, ck2, ck3;
LOCAL rate,total;

// Input Command
INPUT({item1,{ck1,1},item2,{ck2,1},
item3,{ck3,1},rate},
"Check Out",
{"Item1:", "Tax1:", "Item2:", "Tax2:",
"Item3:", "Tax3:", "Rate:"});

// Calculation
rate:=rate/100;
total:=item1*(1+ck1*rate)
+item2*(1+ck2*rate)
+item3*(1+ck3*rate);

// Display total
RETURN total;

END;

```

Example: Analyze the total of three items: \$59.99 subject to sales tax, \$9.99 subject to sales tax, and \$10.00 not subject to sales tax. The sales tax rate is 9%. Turn the first two checkboxes on while leave the last check box unchecked.

Result: \$86.28

REPEAT

ULAM Algorithm: take an integer n . If n is even, divide it by 2. If n is odd, multiply it by 3 and add 1. ULAM counts how many steps it takes to get n to 1.

The REPEAT Structure

The REPEAT loop repeats a set of instructions until a condition is met. Once it is met, the loop is.

```

REPEAT
commands
UNTIL condition;

```

Access: (Tmpl), 3. Loop, 6. REPEAT

The Program ULAM

The ULAM algorithm takes an integer n and tests whether n is divisible by 2 (even). If n is even, divide it by 2. If n is odd, multiply it by 3 and add 1. ULAM counts how many steps it takes to get

n to 1.

```
EXPORT ULAM(N)
BEGIN
LOCAL C:=1, L0:={N};

REPEAT

IF FP(N/2)==0 THEN
N:=N/2;
ELSE
N:=3*N+1;
END;

C:=C+1;
L0:=CONCAT(L0,{N});
UNTIL N==1;

MSGBOX("NO. OF STEPS="+C);
RETURN L0;
END;
```

Examples:

ULAM(5) returns:

Message Box: "NO. OF STEPS=6"

List: {5, 16, 8, 4, 2, 1}

ULAM(22) returns:

Message Box: "NO. OF STEPS=16"

List: {22, 11, 34, 17, 52, 26, 13, 40, 20, 10, 5, 16, 8, 4, 2, 1}

The GETKEY Command

The next section will introduce a super-important command, GETKEY. We will be working with GETKEY over the entire series.

GETKEY: Returns the key code of last key pressed. The key codes are displayed on the table below.

Access: (Cmds), 6. I/O, 5. GETKEY

KEY	Apps	Symb	Plot	Num	Home	Help
Key Code	0	1	6	11	5	3
KEY	View	Menu	Esc	CAS		
Key Code	9	13	4	10		
KEY	↑	←	→	↓		
Key Code	2	7	8	12		
KEY	Vars	Toolbox	Template	x t θ n	a b/c	Backspace

Key Code	14	15	16	17	18	19
KEY	x^y	SIN	COS	TAN	LN	LOG
Key Code	20	21	22	23	24	25
KEY	x^2	+/-	()	,	Enter	
Key Code	26	27	28	29	30	
KEY	EEX	7	8	9	÷	
Key Code	31	32	33	34	35	
KEY	ALPHA	4	5	6	×	
Key Code	36	37	38	39	40	
KEY	Shift	1	2	3	-	
Key Code	41	42	43	44	45	
KEY	On	0	.	(space)	+	
Key Code	46	47	48	49	50	

The Program KEYNO

The person presses key presses. Which each key press, the code returns to the terminal screen. The program terminates when the Enter key is pressed.

```
EXPORT KEYNO()
BEGIN
LOCAL K;
PRINT();
PRINT("Press any key to get its code.");
PRINT("Press Enter to exit.");

REPEAT
K:=GETKEY;
IF K ≥ 0 THEN
PRINT(K);
END;
UNTIL K==30;

END;
```

Tutorial #4: CHOOSE and CASE

The CHOOSE Command

The CHOOSE command Creates a pop up choose box, similar to what you see when you click on a soft menu. There are two syntaxes for CHOOSE:

Simple Syntax (up to 14 options):

CHOOSE(var, "title string", "item 1", "item 2", ... , "item n");

List syntax (infinite amount of items):

CHOOSE(var, "title string", {"item 1", "item 2"});

Choosing item 1 assigns the value of 1 to var, choosing item 2 assigns the value of 2 to var, and so on.

Access: (Cmds), 6. I/O, 9. CHOOSE

The Program TERMVEL

The Program TERMVEL calculates the terminal velocity of an object given its shape and the units to be used (kilograms/meters/seconds or pounds/feet/seconds). Lists are created to store the necessary data.

```
EXPORT TERMVEL()
BEGIN

LOCAL L0:={9.80665,32.174},
L1:={1.225,.0765},
L2:={.47,1.05,1.15,.04},C,K,M,A,T;

CHOOSE(C,"Units","SI","English");

CHOOSE(K,"Type of Object","Sphere","Cube",
"Cylinder","Tear-Shaped");

INPUT({M,A},"Object",
{"M=","A="},{ "Mass", "Surface Area"});

T:=√((2*M*L0(C))/(L1(C)*A*L2(K)));

MSGBOX("Terminal Velocity="+T);
RETURN T;
END;
```

Examples:

Sphere, SI Units, M = .05 kg, A = .0028 m²

Terminal Velocity: $T = 24.6640475387 \text{ m/s}$

Cube, US Units, $M = 1.2 \text{ lb}$, $A = .3403 \text{ ft}^2$

Terminal Velocity: $T = 53.149821209 \text{ ft/s}$

The CASE Structure

The command CASE allows for a group of conditions to test. Think of it as a bunch of "IF-THEN-ELSE" structures, but for that one variable. What CASE also allows is a set of commands if none of the conditions are true, which is the default. The syntax structure:

```
CASE
IF test 1 THEN do if true END;
IF test 2 THEN do if true END;
...
DEFAULT commands END;
```

Access: (Cmds), 2. Branch, 3. CASE

The program AREAC (Area of Circles, Rings, and Sectors) demonstrates the use of a CASE structure.

The Program AREAC

```
EXPORT AREAC()
BEGIN
LOCAL C,R,S,θ,A;
CHOOSE(C,"Areas","1. Circle","2. Ring","3. Sector");

INPUT(R, "Input Radius", "R =");

// Case structure begins
CASE
IF C==1 THEN A:=π*R^2; END;

IF C==2 THEN
INPUT(S, "Small Radius", "r=");
A:=π*(R^2-S^2);
END;

IF C==3
INPUT(θ, "Angle", "θ=");
// Assume you are in the correct angle mode
IF HAngle==1 THEN
// Test Angle Mode
θ:=θ*π/180;
END;
A:=θ*R^2/2;
```

```
END;
```

```
END;
```

```
MSGBOX("Area is "+A);
```

```
RETURN A;
```

```
END;
```

Examples:

$R = 2.5$, $r = 1.5$, $\theta = \pi/4$ radians or 45°

Circle: 19.6349540849

Ring: 12.5663706144

Sector: 2.45436926062

Tutorial #5: STARTAPP, STARTVIEW, CHECK, UNCHECK, RGB, HAngle

We can use the HP Prime's PPL to work with different Applications and use their functions. In this tutorial, we will work with switching the HP Prime to different apps. This will involve system variables that are designated for certain types of variables.

System Equation Variables

The HP Prime designates system equation variables for certain types of equations. They are:

Type (App Name)	Designation: # is from 0 to 9	Independent Variable
Function	F#	X
Polar	R#	θ
Parametric	X#, Y#	T
Sequence	U#	N, N-1, N-2
Advanced Graphing	V#	X, Y
Graph 3D	FZ#	X, Y
Solve	E#	

For example F1 refers to Function F1(X), V2 refers to open statement V2(X,Y), and so on. Solve equations are used with the Solve.SOLVE function.

Defining Equations from Home and Inside Programs

Defining equations this way leaves them unchecked. If you want them plotted or accessed in Num View, you will need to check them.

Examples:

F1:="2*X^3" stores the function $f(x) = 2x^3$ in Function 1.

R5:="A*SIN(θ)" stores the polar function $r(\theta) = A \sin(\theta)$ in Polar Function 5, with A being what value stored in it.

The STARTAPP Command

Syntax:

```
STARTAPP(application name in quotes);
```

Starts the named App. The calculator points the screen to the default view (Plot, Symb, Num).

Access: (Cmds), 4. App Functions, 2. STARTAPP

For example, STARTAPP("Polar") starts the Polar app.

The CHECK and UNCHECK Commands

Checks and unchecks specific equation or function (0-9) in the current app. For example, if you are in the Function app, CHECK(1) activates F1. As you should expect, UNCHECK(1) turns F1 off.

What does CHECK and UNCHECK affect?

1. Whether a function is plotted in Plot view.
2. Whether a function is analyzed in Num view.

Please note that CHECK and UNCHECK are *App Specific* functions. App specific functions are functions that belong to a specific app. Unless you are working the current app, CHECK and UNCHECK will proceed the syntax:

Appname.CHECK(n)

Appname.UNCHECK(n)

It is always safer to use the CHECK and UNCHECK commands with the *Appname*. (include the period) prefix.

Access: [Toolbox], (App), select your app (CHECK and UNCHECK, if available, are in various places in the menu)

Examples:

Function.CHECK(1) checks and turns function F1(X) on.

Solve.UNCHECK(0) unchecks solve equation E0.

The STARTVIEW Command

The STARTVIEW command instructs the HP Prime to go to a certain view. It has two arguments, the view number and a redraw number.

Common view numbers include (not all inclusive):

- 2 = Modes screen
- 1 = Home
- 0 = Symbolic (Symb)
- 1 = Plot
- 2 = Numeric (Num)
- 3 = Symbolic Setup

4 = Plot Setup
 5 = Numeric Setup
 6 = App Information
 7 = The Views Key
 8 = first special view
 9 = second special view
 Etc..

The redraw number is either 0 or non-zero. 0 does not redraw the screen, anything else does.
 The redraw number is optional.

Syntax: `STARTVIEW(view number, [redraw number])`

Access: (Cmds), 4. App Functions, 1. STARTVIEW

Designating Colors with RGB and Hex Codes

Returns an integer code pertaining to a color's RGB code. This is super useful for drawing and text writing.

Syntax: `RGB(red, green, blue, alpha)`

Red: Intensity of Red, 0-255

Green: Intensity of Green, 0-255

Blue: Intensity of Blue, 0-255

Alpha: (optional) Opacity (up to 128).

Access: (Cmds), 2. Drawing, 4. RGB

You can also designate colors by hexadecimal code using the format `#RRGGBBh`. RR is for red (00 – FF), green (00 – FF), and blue (00 – FF). The # prefix and h suffix tells the HP Prime that this is a hexadecimal integer.

Red: `#FF0000h`

Green: `#00FF00h`

Blue: `#0000FFh`

Here is a list of colors with a hex conversion table. White, silver, black, red, magenta, violet, blue, lime green, cyan, teal, gray, maroon, and yellow are exact values. Others are approximate values.

White	255/255/255	Pacific Blue	0/160/192	DEC:HEX
Silver	192/192/192	Teal	0/128/128	08: 08
Gray	128/128/128	Cyan/Aqua	0/255/255	16: 10
Black	0/0/0	Fir Green	0/48/0	32: 20
Red	255/0/0	Office Green	0/128/0	48: 30
Magenta	255/0/255	Shamrock	0/160/96	64: 40
Cardinal	208/48/32	Apple	80/176/64	80: 50
Coral	240/128/112	Lime Green	0/255/0	96: 60
Pink	255/192/192	Chocolate	64/8/8	112: 70
Indigo	64/0/128	Brown	144/80/0	128: 80

Purple	128/0/128	Orange	255/128/0	144: 90
Violet	128/0/255	Olive	128/128/0	160: A0
Lavender	192/192/255	Tangerine	255/208/0	176: B0
Navy Blue	0/0/128	Yellow	255/255/0	192: C0
Blue	0/0/255	Khaki	240/224/144	208: D0
Denim	16/96/192	Tan	208/176/144	224: E0
Lochmara	0/112/192	Maroon	128/0/0	240: F0
Sky Blue	128/208/255	Snow White	255/240/240	255: FF

The Color of a Graph

Check the color of a graph by using the syntax:

`F# (COLOR) **`

This returns a hexadecimal value for the color. This applies to any of the equation variables F#, R#, (X# and Y# will have the same value), U#, V#, and E#. There is a list of two colors for FZ# (3D Graphing).

To change the color of a graph, use the syntax:

`F# (COLOR) :=RGB (red,blue,green, [alpha]) **`

`F# (COLOR) :=#RRGGBBh **`

Example:

`F8(COLOR):=RGB(0,0,255)` makes the function F8 plot in [blue](#).

The System Variable HAngle

The system variable HAngle (Home Angle mode) has three values:

HAngle = 0 for Radians mode

HAngle = 1 for Degrees mode

HAngle = 2 for Gradians mode

The value of HAngle can be recalled and values can be stored in HAngle.

Access: [Vars], (Home), 6. Settings, 1. HAngle

The Program CONIC

The program CONIC draws the general equation

$$Ax^2 + By^2 + Cxy + Dx + Ey + F = 0$$

You can choose the color how the conic section is plotted, from [red](#), [blue](#), [orange](#), and [green](#). (Game show enthusiasts, I hope you note the order of the colors I listed... ;)).

```

EXPORT CONIC()
BEGIN

// Initialize variables
LOCAL cr, cg, cb, I;

// Input box with default and reset
// values set at 0, help text is an
// empty list
INPUT({A,B,C,D,E,F},
"Ax^2+By^2+Cxy+Dx+Ey+F", { },
{0,0,0,0,0,0},{0,0,0,0,0,0});

// Colors
CHOOSE(I, "Choose a Color",
"Red", "Blue", "Orange", "Green");
// Red values list
cr:={255,0,255,0};
// Green values list
cg:={0,0,127,255};
// Blue values list
cb:={0,255,0,0};

// Switch to Advanced Graphing app
STARTAPP("Advanced Graphing");

// Set equation V1, colors, check
// Here we are in the Advanced Graphing
// app, so no designation is necessary
// for CHECK
V1:="A*X^2+B*Y^2+C*X*Y+D*X+E*Y+F=0";
V1(COLOR):=RGB(cr(I),cg(I),cb(I));
CHECK(1);

// Plot View
STARTVIEW(1,1);
END;

```

The Program PROJ13

The program PROJ13 calculates range and height of a projectile, and plots its path. The program sets the mode into Degrees (HAngle=1) and the calculator to the Parametric app.

Equations:

$$x = V * \cos \theta * t$$

$$y = V * \sin \theta * t - .5 * g * t^2$$

Where:

V = initial velocity

θ = initial degree of flight

g = Earth gravitation constant (9.80665 m/s², \approx 32.17404 ft/s²)

Air resistance is not factored, so we are dealing with ideal conditions. How much the projectile represents reality varies, where factors include the object being projected, the temperature and pressure of the air, and the weather.

```
EXPORT PROJ13()
BEGIN
LOCAL M, str;
// V, G,  $\theta$  are global
// Degrees
HAngle:=1;

CHOOSE(M, "Units", "SI", "US");
IF M==1 THEN
str:="m";
G:=9.80665;
ELSE
str:="ft";
G:=32.17404;
END;

INPUT({V,  $\theta$ }, "Data",
{"V:", " $\theta$ :"},
{"Initial Velocity in "+str+"/s",
"Initial Angle in Degrees"});

X1:="V*COS( $\theta$ )*T";
Y1:="V*SIN( $\theta$ )*T-.5*G*T^2";

STARTAPP("Parametric");
CHECK(1);
// Adjust Window
Xmin:=0;
// Range
Xmax:=V^2/G*SIN(2* $\theta$ );
Ymin:=0;
// Height
Ymax:=(V^2*SIN( $\theta$ )^2)/(2*G);
MSGBOX("Range: "+Xmax+" "+str+", "
+", Height: "+Ymax+" "+str);
// Plot View
STARTVIEW(1,1);
END;
```

Tutorial #6: Subroutines

How Subroutines Work

This session will show how routines work in PPL. Generally, subroutines have be declared *before* the main program. Declaration is important. The details of the subroutines are *after* the main program.

Definitely take a look at the example programs to get a better understanding.

SUB Routines for HP Prime

General Syntax:

```
sub(); //declare subroutines

EXPORT main()
BEGIN
  commands go here, including sub()
END;

sub()
BEGIN
  commands go here
END;
```

The Program SUBEXAM

This is just a demonstration of how sub routines work. This program calculates one of two values:

If A is positive, then the program evaluates A. If not, the program values B instead. Where:

$$A = 2(x-y)/\Phi + xy$$
$$B = \Phi^2$$

$$\text{and } \Phi = 2e^{(x+y)} - e^{(x-y)} - e^{(y-x)}$$

We will use Φ as the subroutine.

```
SUB1 ();

EXPORT SUBEXAM(X, Y)
BEGIN
  LOCAL A, B;
  A := (2 * (Y - X)) / SUB1 (X, Y) + X * Y;
  B := (SUB1 (X, Y)) ^ 2;
```

```
IF A>B THEN
RETURN A;
ELSE
RETURN B;
END;
END;
```

```
SUB1 (X,Y)
BEGIN
RETURN  $2 * e^{(X+Y)} - e^{(X-Y)} - e^{(Y-X)}$  ;
END;
```

Examples:

SUBEXAM(-4, 1) returns 21998.918189

SUBEXAM(2,3) returns 86283.2797974

SUBEXAM(-5,-6) returns 30.648061288

SUBEXAM(2,-3) returns 21810.6046664

Tutorial #7: Graphics Screen, TEXTOUT

The Graphics Screen: Cartesian vs Pixel

OK this section we will start covering some of the graphics features of the HP Prime Programming Language. We touched on graphics a bit when we used STARTAPP and STARTVIEW to call up the Plot screen of certain apps (Function, Parametric, Polar, Sequence, Advanced Graphing, Graph 3D).

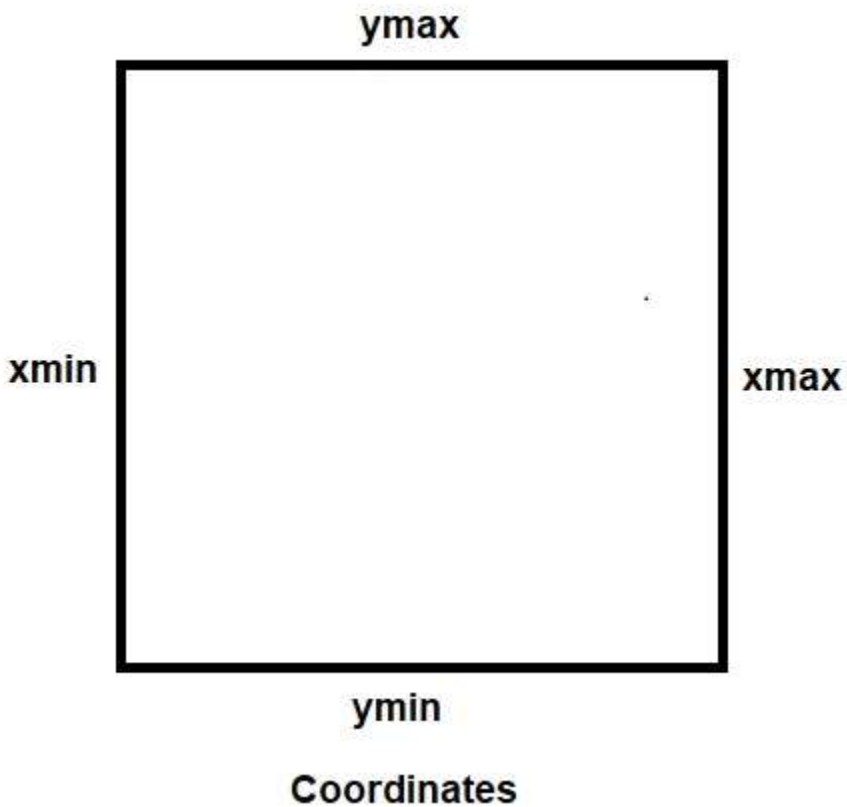
This time we are going to use drawing commands that can be used in any HP Prime app. In a sense we are creating a graphic object (GROB). The HP Prime allows for ten graphic objects, named G0 - G9. For this tutorial series, (unless specified) I am always going to use the default GROB, G0. This makes typing commands much easier.

Each graphic object operates in either one of two coordinate systems: Cartesian and pixel.

The features of the Cartesian system (x,y) are these:

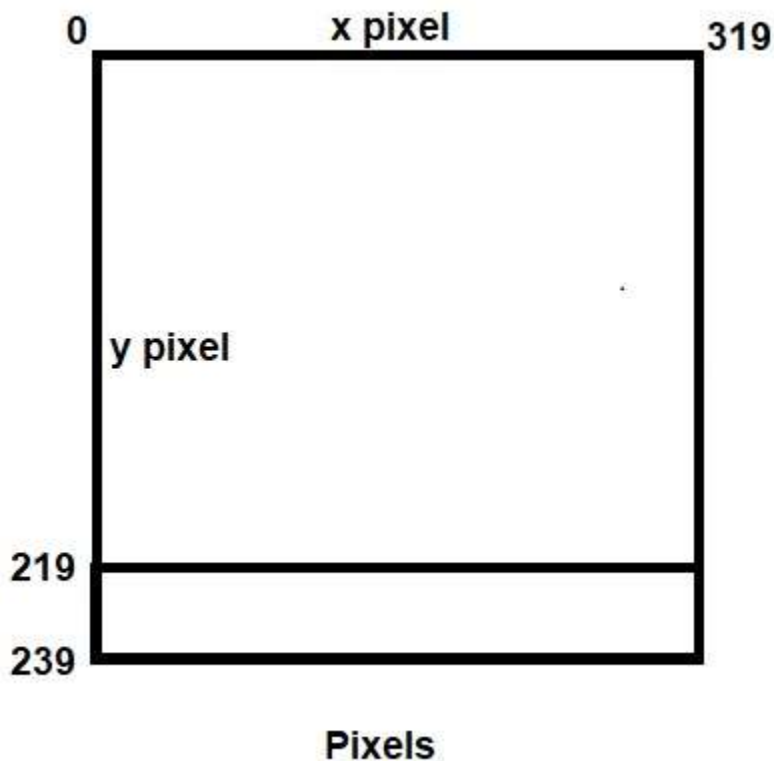
1. The end points depend on the Plot variables Xmin, Xmax, Ymin, and Ymax.
2. The system is familiar, having x increasing as we move to the right and y increasing as we move up.
3. The trade is that some (very few) drawing commands don't accommodate the Cartesian system. An example is the ARC command, which requires the radius to be in pixels.

Next is a map of the Cartesian system:



The Pixel System (x,y):

1. The boundaries are **fixed**. The pixel (0,0) is the top left hand corner, the pixel (318, 218) is the lower right hand corner.
2. The value of x still increases as we go to the right. However, y increases as we go down, opposite of the Cartesian system. On the other hand, x and y are always non-negative.



Note that the y pixels from 219 to 239 are allocated for the soft touch menu.

The Drawing Commands

The HP Prime has two sets of drawing commands: one for the Cartesian system and one for the Pixel system. All commands for the Pixel system will have a "_P" suffix attached. For example:

LINE draws a line using Cartesian coordinates, while LINE_P draws a line using Pixel coordinates.

General Access: (in the Programming Editor)

Drawing Commands for the Pixel system: (Cmds), 2. Drawing, 6. Pixels

Drawing Commands for the Cartesian system: (Cmds), 2. Drawing, 7. Cartésian

Clearing the GROB screen

To clear the GROB screen, we will simply type `RECT()`. This wipes the screen, leaving it white. It is necessary to do this at least at the beginning of each program containing drawing commands. In a sense, `RECT()` is similar to `PRINT()`.

Hint: To paint an entire screen a specific color, use `RECT (color)`.

Displaying the Graphics Screen with FREEZE and WAIT

It is not enough to type the drawing commands. We need a command to tell the HP Prime to show the graphics. Two ways to do it are:

FREEZE: This does exactly what it says, freezes the screen. To exit, tap the screen or press ESC. Pressing Enter will re-execute the program.

Access: (Cmds), 2. Drawing, 1. FREEZE

WAIT(0): This freezes the screen for an indefinite amount of time. However, pressing any button will cause the program to continue. Of course, if the last END is followed by WAIT(0), the program terminates.

Of course, you can use WAIT(n) to make the calculator wait n seconds before executing the next step.

Access: (Cmds), 6. I/O, 3. WAIT

The Commands TEXTOUT and TEXTOUT_P

TEXTOUT and TEXTOUT_P inserts text on a graphics object using Cartesian and Pixel coordinates, respectively. They are also at the bottom of the Cartesian and Pixel Drawing sub menus, respectively. (Use either the [x,t,θ,n] button or the up button followed by Enter).

Full Syntax (starred commands are optional):

Cartesian:

```
TEXTOUT(text, GROB*, x, y, font size*, text color*, width*, background color*)
```

Pixel:

```
TEXTOUT_P(text, GROB*, x, y, font size*, text color*, width*, background color*)
```

text: The text to be written. It can be a string, results, calculations, or any combination.

GROB*: Graphic Object G0 through G9 to be used. If left out, G0 is used.

x: x coordinate

y: y coordinate

font size*: The text font's size code. Must be used if you want text to be a color other than black. Optional. Default is the current size set by Home Settings.

0: Current font size as set by Home Settings screen.

1: Size 10 font

2: Size 12 font

3: Size 14 font

4: Size 16 font

5: Size 18 font

6: Size 20 font

7: Size 22 font

text color*: The color of the text. Use of the RGB command is advised. Optional. Default color is black.

width*: Length of the background box of the text. Optional. I usually don't use this argument.

background color*: Color of the background box. Optional. I usually don't use this argument.

Simplified Syntaxes:

Black text at default font size:

```
TEXTOUT(text, x, y)
```

```
TEXTOUT_P(text, x, y)
```

Colored text at a set font size:

```
TEXTOUT(text, x, y, size code, color)
```

```
TEXTOUT_P(text, x, y, size code, color)
```

The Program SNOWFLAKE

With all this, we finally get to some programming. Typically during winter, it snows a lot. Let's use TEXTOUT_P to draw snowflakes. I am going to use symbolize the snowflake by the asterisk, the symbol of multiplication in programming. [*] types *.

SNOWFLAKE takes one argument, which is the number of snowflakes to be drawn.

Note: Take note the order of the commands. The order regarding where to draw and generate random numbers is important to get the results you want.

```
EXPORT SNOWFLAKE(N)
BEGIN
  LOCAL X,Y,Z,I,L0;
  L0:={RGB(0,0,255),RGB(178,255,255),
  RGB(30,144,255),RGB(0,255,255)};
  \\ blue, light blue, dodger blue, cyan

  RECT();

  FOR I FROM 1 TO N DO
  X:=RANDINT(0,304); \\ save some room since text takes pixels
```

```
Y:=RANDINT(0,208);  
Z:=RANDINT(1,4);  
Z:=L0(Z);  
TEXTOUT_P("*",X,Y,2,Z);  
END;
```

```
FREEZE;  
END;
```

Tutorial #8: LINE, FILLPOLY, ARC

All the example programs shown will use pixel coordinates.

The LINE Command

The LINE command draws a line from one coordinate to the other. The general syntax is:

Cartesian:

```
LINE(x1, y1, x2, y2, color*)
```

Access: (Cmds), 2. Drawing, 7. Cartésian, 8. LINE

Pixels:

```
LINE_P(x1, y1, x2, y2, color*)
```

Access: (Cmds), 2. Drawing, 6. Pixels, 8. LINE_P

The Program DRAWHOUSE

DRAWHOUSE demonstrates the LINE_P command to draw a house, the base is brown (#905000h), and the roof is maroon (#800000h).

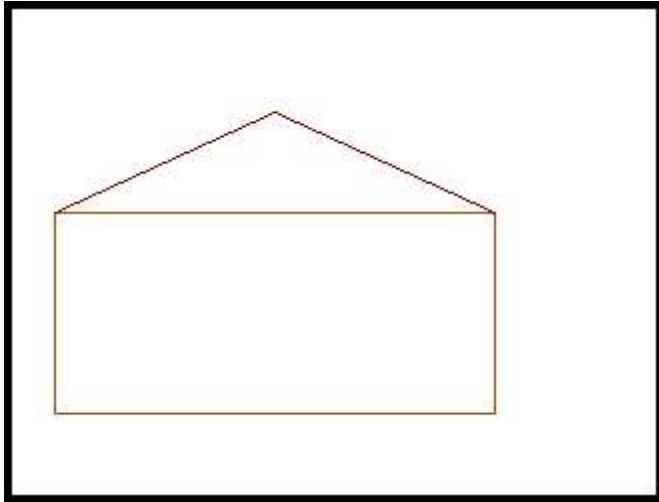
```
EXPORT DRAWHOUSE()
BEGIN

// Clear the screen
RECT();

// Draw the house
LINE_P(20,100,20,200,#905000h);
LINE_P(20,200,240,200,#905000h);
LINE_P(240,200,240,100,#905000h);
LINE_P(240,100,20,100,#905000h);
LINE_P(20,100,130,50,#800000h);
LINE_P(130,50,240,100,#800000h);

// Show the screen
WAIT(0);

END;
```



Drawing Polygons with the FILLPOLY Command

The FILLPOLY command draws a filled polygon. The general syntax is:

Cartesian:

```
FILLPOLY(list of points, color, [alpha])
```

Access: (Cmds), 2. Drawing, 7. Cartésian, E. FILLPOLY

Pixels:

```
FILLPOLY_P(list of pixel pairs, colors, [alpha])
```

Access: (Cmds), 2. Drawing, 6. Pixels, E. FILLPOLY_P

The order of the points are important.

The Program PENT

The program DRAWPENT draws a five sided polygon in pink (#FFC0C0h) on an indigo screen (#400080h).

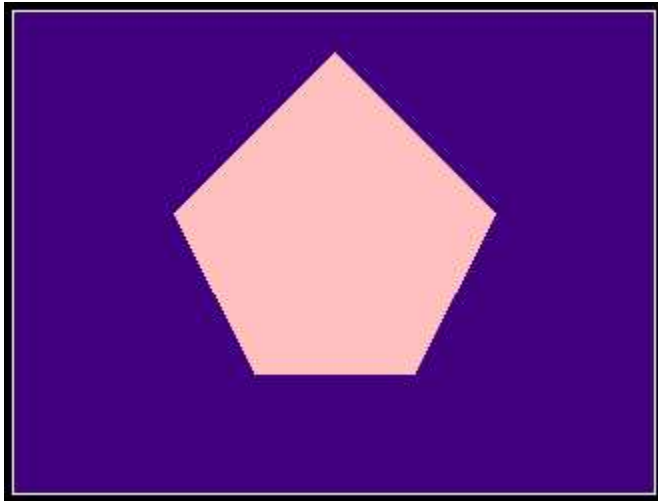
```
EXPORT DRAWPENT()  
BEGIN  
  
// Clear the screen  
// Indigo background  
RECT(#400080h);
```



```
// Draw a five sided polygon in pink
FILLPOLY_P({(80,100),(160,20),(240,100),
(200,180),(120,180)},#FFC0C0h);

// Show the screen
WAIT(0);

END;
```



The ARC Command

The ARC command is used to draw arcs, ellipses, and circles. What is important to remember is that the radius with the ARC command will be in pixels regardless of which version of the ARC command that is used. The syntax for ARC_P will be presented here (the syntax for ARC will be similar).

Access: (Cmds), 2. Drawing, 6. Pixels, 1. ARC_P

Drawing Circles with ARC

Draw a circle:

```
ARC_P(center x, center y, radius, color)
```

Draw a filled circle:

```
ARC_P(center x, center y, radius, {border color, filled color})
```

Drawing Ellipses with ARC

Draw a circle:

```
ARC_P(center x, center y, {x semi-axis radius, y semi-axis radius},  
color)
```

Draw a filled circle:

```
ARC_P(center x, center y, {x semi-axis radius, y semi-axis radius},  
{border color, filled color})
```

Drawing Partial Arcs with ARC

Draw a partial arc between two angles (think unit circle)

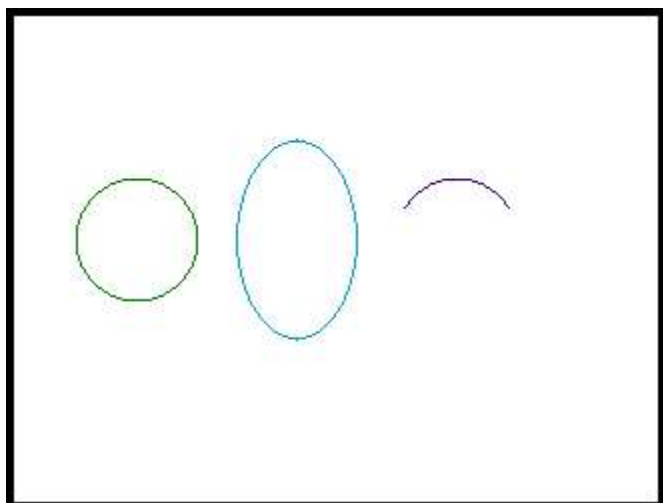
```
ARC_P(center x, center y, radius, angle 1, angle 2, color)
```

Filled arc:

```
ARC_P(center x, center y, radius, angle 1, angle 2, {border color,  
filled color})
```

The Program DRAWARCS

```
EXPORT DRAWARCS()  
BEGIN  
  
// Clear the drawing screen  
RECT();  
  
// Draw a circle, office green  
ARC_P(60,110,30,#008000h);  
  
// Draw an ellipse, pacific blue  
ARC_P(140,110,{30,50},#00A0C0h);  
  
// Draw an arc from 30 to 150 degrees,  
// indigo  
HAngle:=1;  
ARC_P(220,110,30,30,150,#400080h);  
  
// Show the screen  
WAIT(0);  
  
END;
```



Tutorial #9: Numerical Calculus with the Template Key

We can use the template key, which is located on the top row of gray keys, third key from the left (from here on in marked as [Template]).

Numerical Derivatives

Syntax: ∂ (f(variable), variable = value)

Access: [Template], 1st row, 4th column

Definite Integrals

Syntax: \int (f(variable), variable, lower limit, upper limit)

Access: [Template], 2nd row, 4th column

Double Definite Integrals

Syntax: \int (\int (f(X,Y), X, A, B), Y, C, D)

This computes $\int \int f(x,y) dx dy$

Where X and Y represent the first and second variable, A and B are limits for the first variable, and C and D are limits for the second variable.

Summation

Syntax: Σ (f(variable), variable, lower limit, upper limit)

Access: [Template], 3rd row, 3rd column

With this function, the step is assumed to be 1, with both the lower and upper limits being integers.

Substitution

One variable:

Syntax: expression|variable = value

Multiple variables:

Syntax: expression|{variable 1 = value 1, variable 2 = value 2, and so on}

Access: [Template], 1st row, 3 column

The Program CALCDEMO

The program CALCDEMO demonstrates the use of the five template functions as discussed in this section. The terminal is used to print the results and the calculator is set to Radians mode.

```
EXPORT CALCDEMO()
BEGIN
// Demonstration of Template functions

// Clear Print Terminal
PRINT();

// Change angle to Radians
HAngle:=0;

// Derivatives
PRINT("d/dx sin x at x =  $\pi/4$ ");
PRINT( $\partial(\text{SIN}(X), X=\pi/4)$ );

// Integrals
PRINT(" $\int(\sin x \, dx)$  from x=1 to x=3");
PRINT( $\int(\text{SIN}(X), X, 1, 3)$ );

// Double Integrals
PRINT(" $\int(\int(x^2*y \, dx)dy)$ , x=[1,2], y=[0,4]");
PRINT( $\int(\int(X^2*Y, X, 1, 2), Y, 0, 4)$ );

// Summation
PRINT(" $\Sigma(n^3)$  from n=1 to n=12");
PRINT( $\Sigma(N^3, N, 1, 12)$ );

// Substitution
PRINT("Calculate  $3*a+5$  when  $a=-2$ ");
PRINT( $3*A+5|A=-2$ );

END;
```

Results should be:

0.707106781186
1.53029480247
18.6666666667
6084
-1

