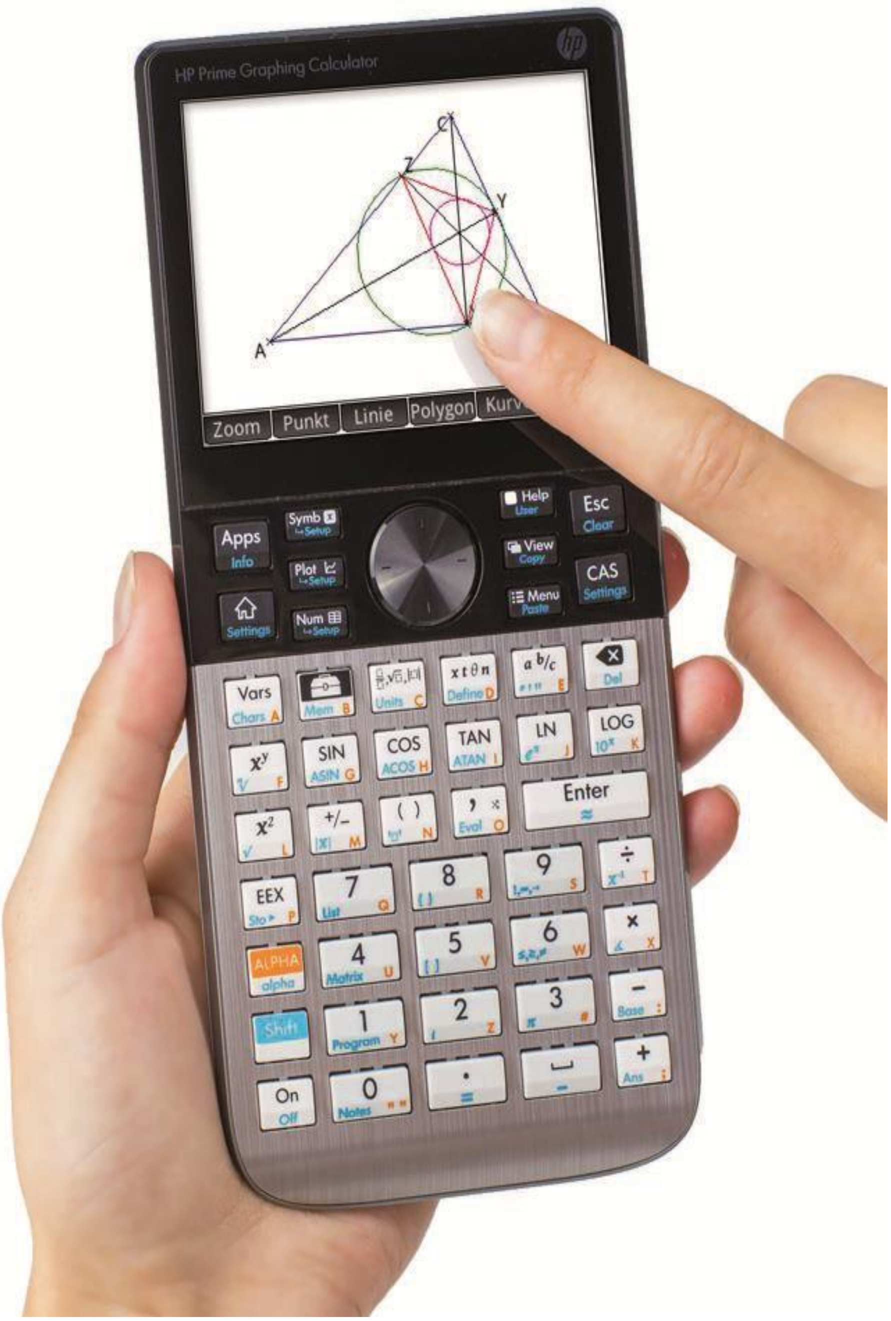
Python Activities Book

HP Prime



Created by Neil Streeter June 2025

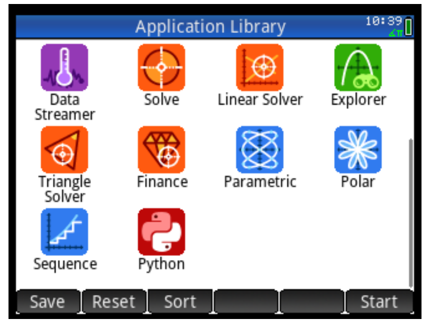
Python Programming

Contents

HP Prime Graphing Calculator

1. Getting started with Python on the HP Prime

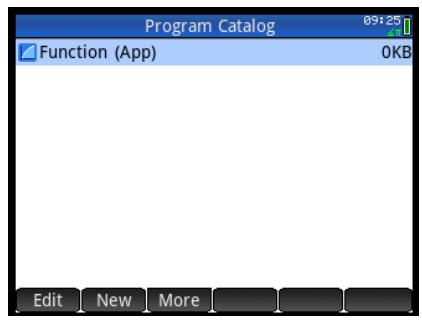
Since firmware update version Beta 2.1.15048 the HP Prime Graphing calculator has had an ability to program in Python and the ability to create Python based applications. Though not a full implementation of Python - Micro python is closely based on Python 3.4 with some hardware specific and library differences.



The HP-Prime uses micro python (You should be able to find all the documentation at https://micropython.org/). The closest version for reference is https://docs.micropython.org/en/v1.9.4/ however while additional functions have been added to take advantage of the calculators built in capabilities and some functions such as the control of I/O ports are not included.

There are two methods for creating Python programs: -

1. By using or creating a copy of the Python application.
2. By creating a new program using the Shift 1 Program Y keys.

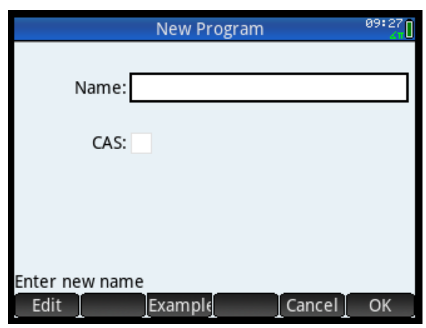


These require slightly different approaches.

For both we will need to enter the Python script, but in the program mode it will need to be placed within a HPL wrapper, in simple terms some code that tells the Prime to interpret the code as Python.

2. Creating Python Programs

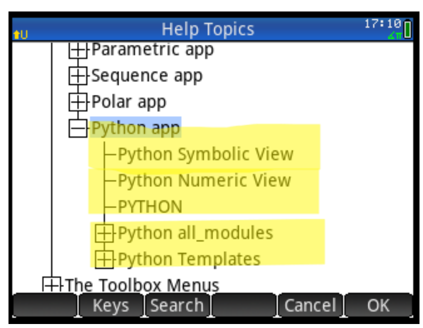
Start by pressing Shift 1 Program Y to access the programs screen.



Then Select New from the options at the bottom of the display. This is the same process that you would use to create a HPPL program.

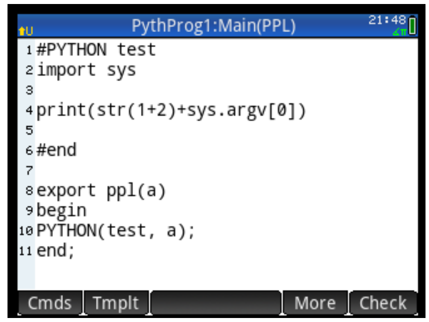
There is no direct help for Python available in the program view; to get help you need to highlight the Python App, then press the Help User button. If you then click on the TREE menu it will open at the Python App help.

The Python tree gives details of how to create the initial wrapper needed for Python to work under HPPL or how Python can be called directly from within PPL using the PYTHON("script",[params]) syntax.



It is well worth reading this help file which I will refer to as we go through this tutorial.

After entering a name for the program, you will be presented with the program entry screen. In this case I have called the program PythProg1.



There are several ways to proceed from here outlined in help. Every program that is not based on a Python App needs to be embedded in a PPL wrapper.

The wrapper will embed the Python code into a script that can be called with the HPPL command PYTHON("script", [params]) within the PPL section to run a Python script.

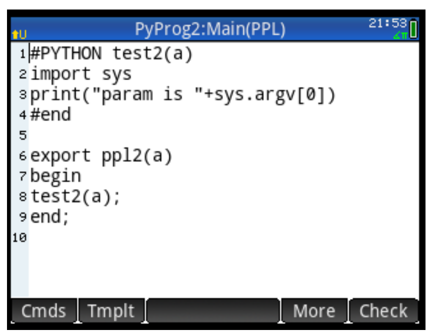
For example:

```
#PYTHON test
import sys
//Put your python code between these two comments
//Beginning of Python code
print(str(1+2)+sys.argv[0])
//End of Python code
#end

export ppl(a)
begin
PYTHON(test, a);
end;
```

A second form executes the python script but without the PYTHON command by using a HPPL wrapper to call Python code from PPL using the name you have given it.

For example:

```
#PYTHON test2,(a)
IMPORT sys
PRINT("param is "+sys.argv[0])
#END

EXPORT ppl2(a)
BEGIN
test2(a);
END;
```



In this example code was entered using Visual Studio Code selecting the HP PL script support extension which appears to include the python syntax, but we will come to that shortly for now let's take a step back.

When you initially create the program after naming it you will get the default PPL structure. Press Shift and Clear (ESC key) to clear this text.

So let's have a look at the PPL wrapper.

```
#PYTHON name        ← Name this as whatever you want
                    ← Place your Python code here
#end

EXPORT PyProg2()    ←This is the name it will appear as
BEGIN
 PYTHON(name);      ←Call the python code by name
END;
```

A note on code entry.

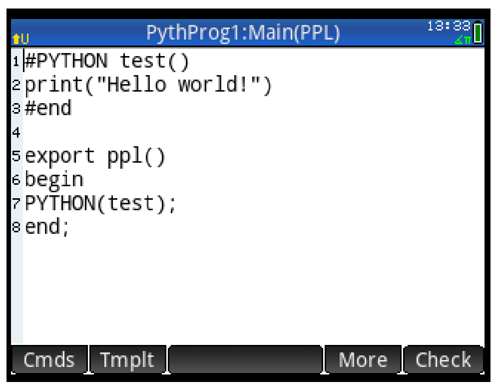
Entering code can be done directly into the calculator on the program entry screen but can be a bit tedious as the keyboard doesn't lend itself to typing. There are a few options available that make this easier.

1. Use the HP Connectivity kit
2. Use Microsoft Visual Studio Code with the HPPL extension
3. Use Notepad and cut and paste the code into a virtual HP Prime calculator.

Options 2 and 3 still require the connectivity kit to transfer the code to the calculator, whether the virtual or actual device. Using VS Code has the advantage of syntax checking and colourisation of Keywords.

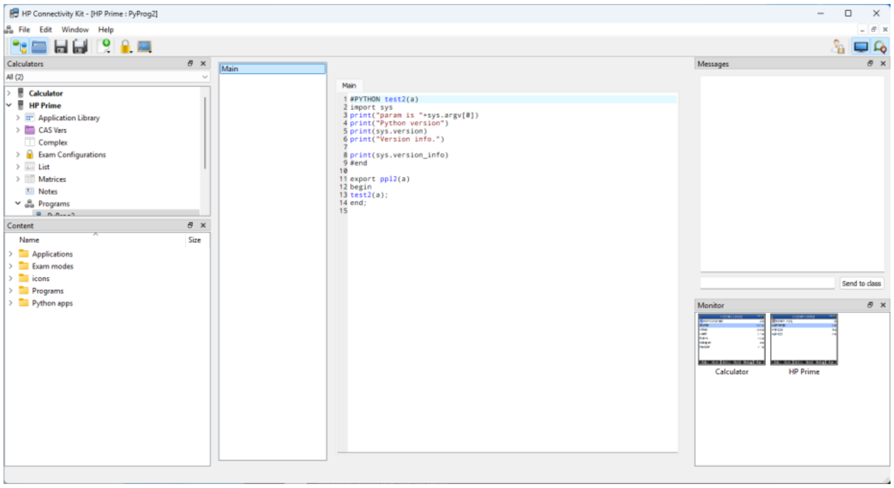
Let's recreate the classic "Hello world" one liner – all that you have to do is embed the Python code for this between the #PYTHON and #END tags.

```
#PYTHON name
PRINT("Hello world!")      ← Place your Python code here
#END

EXPORT PyProg2()
BEGIN
 PYTHON(name);
END;
```



VS Code has significant benefits including Python 3.4 support and Co-Pilot which can help building the code.

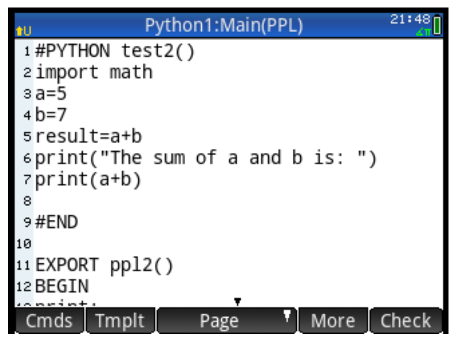
The following shows the same program viewed from within the HP Connectivity Kit and is where you would paste the copied code from VS Code.



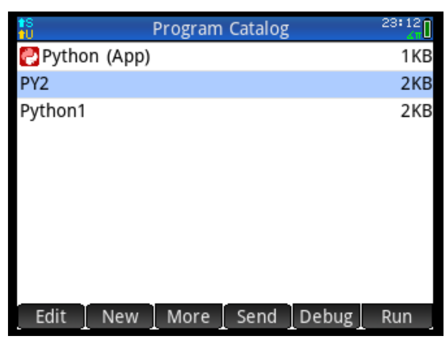
We will cover the tools in a bit more depth further on, but for now just understand, whatever tool you use you copy the code and paste it into the connectivity kit.

3. Python language on the HP Prime

Ok so let’s have a look at Python by examples. First let’s have the program create two variables a and b and add them together and print the answer.




```
#PYTHON test2()
import math
a=5
b=7
result=a+b
print("The sum of a and b is: ")
print(a+b)

#END

EXPORT ppl2()
BEGIN
print;
test2();
END;
```



Use ESC to get to the list of programs, select the program from the list and then click on run.

We can make this more interactive by prompting for the values a and b


```
#PYTHON test2()
a=input("Enter a: ")
print(end = '\n')
b=input("Enter b: ")

print(end = '\n')

print("The sum of a and b is: ")
result = int(a) + int(b)
print(result)

#END

EXPORT ppl2()
BEGIN
print;
  test2();
END;
```

This now prompts you to enter two numbers a and b it adds them together and prints the sum of the two numbers. If you don't cast the variables as type int, then it will assume they are of type string and concatenate the two figures together.

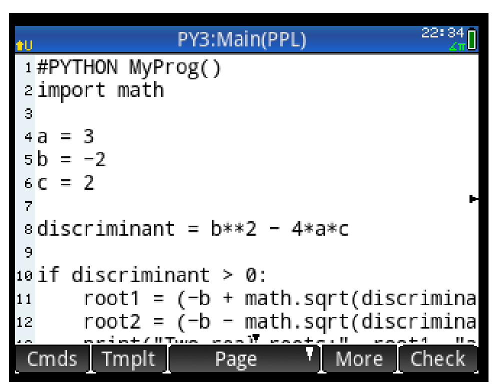
To illustrate the alternative syntax for calling the python script the HPPL part of the program could also be written as follows explicitly using the Python key word.

```
EXPORT ppl2()
BEGIN
print;
PYTHON(test2);
END;
```

So let's add something a bit more complex, in this example we will import the Math library and solve a quadratic of the form

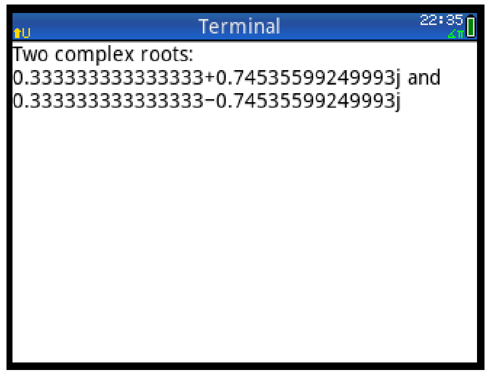
$$ax^2 + bx + c = 0\,,$$

For this example, we will set the values of a b and c in the code.



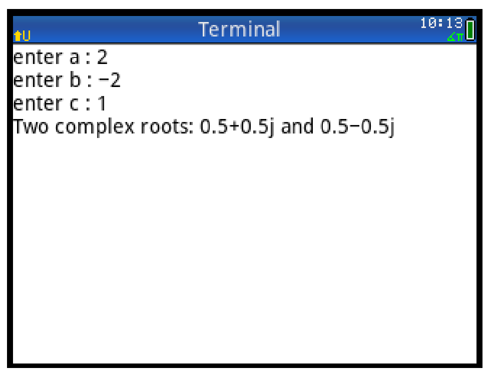
```
#PYTHON MyProg()
import math
a = 3
b = -2
c = 2

discriminant = b**2 - 4*a*c

if discriminant > 0:
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print("Two real roots:", root1, "and", root2)
elif discriminant == 0:
    root = -b / (2*a)
    print("One real root:", root)
else:
    real_part = -b / (2*a)
    imag_part = math.sqrt(-discriminant) / (2*a)
    print("Two complex roots: {}+{}j and {}-
{}j".format(real_part, imag_part, real_part, imag_part))
#END

EXPORT PY3()
BEGIN
  Python(MyProg());
END;
```

The results are displayed in the terminal screen accessed by pressing the Num key.



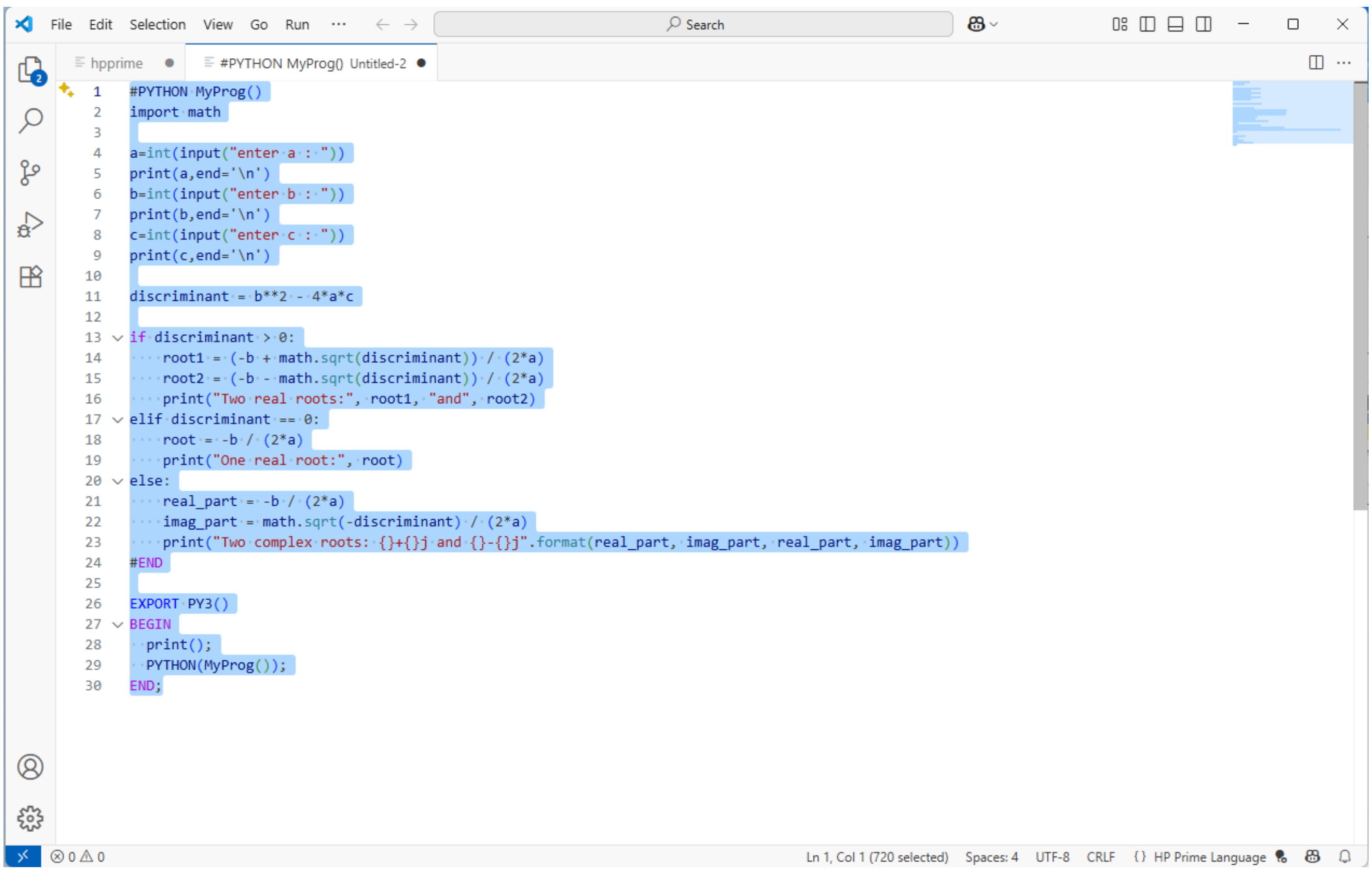
Ok let’s change this so that you can enter the values of a, b, and c.

```
#PYTHON MyProg()
import math

a=int(input("enter a : "))
print(a,end='\n')
b=int(input("enter b : "))
print(b,end='\n')
c=int(input("enter c : "))
print(c,end='\n')

discriminant = b**2 - 4*a*c

if discriminant > 0:
    root1 = (-b + math.sqrt(discriminant)) / (2*a)
    root2 = (-b - math.sqrt(discriminant)) / (2*a)
    print("Two real roots:", root1, "and", root2)
elif discriminant == 0:
```

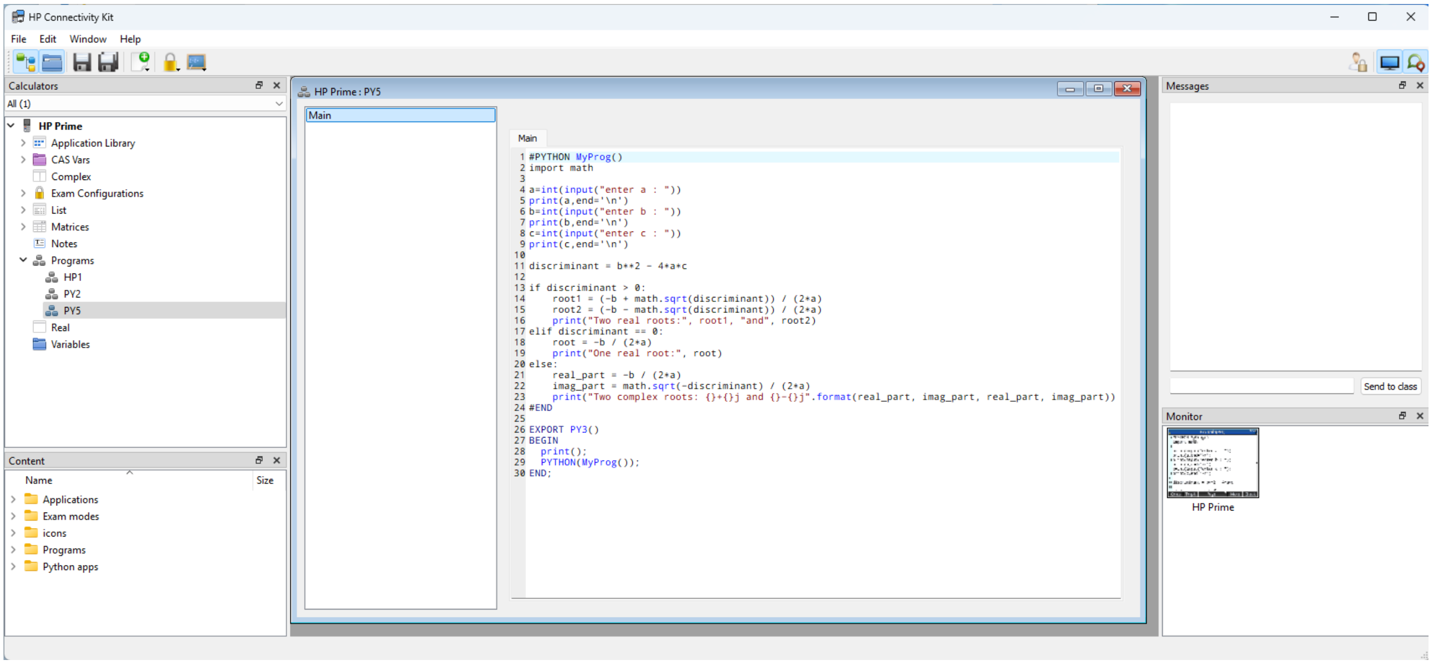


```
    root = -b / (2*a)
    print("One real root:", root)
else:
    real_part = -b / (2*a)
    imag_part = math.sqrt(-discriminant) / (2*a)
    print("Two complex roots: {}+{}j and {}-
{}j".format(real_part, imag_part, real_part, imag_part))
#END

EXPORT PY3()
BEGIN
  print();
  PYTHON(MyProg());
END;
```

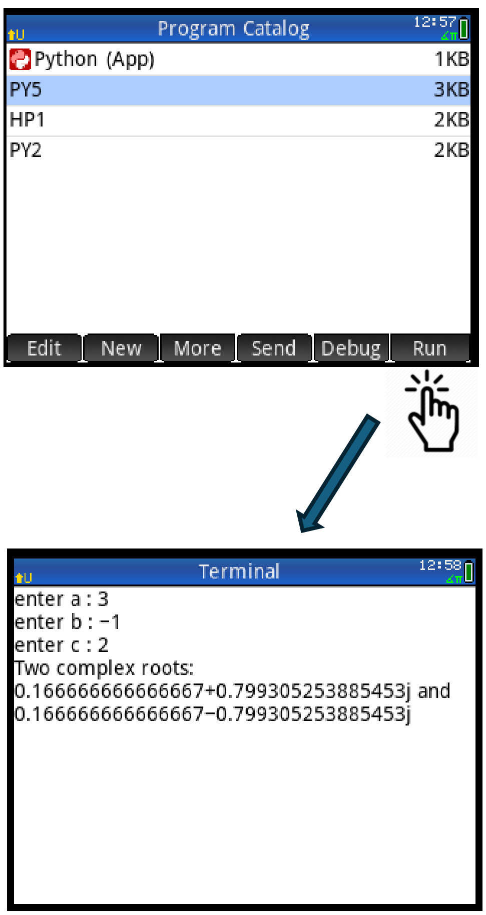
Below is the code being edited in Microsoft Visual studio code. It provides colour coding syntax checking, co-pilot for VS Code and auto-indentation.

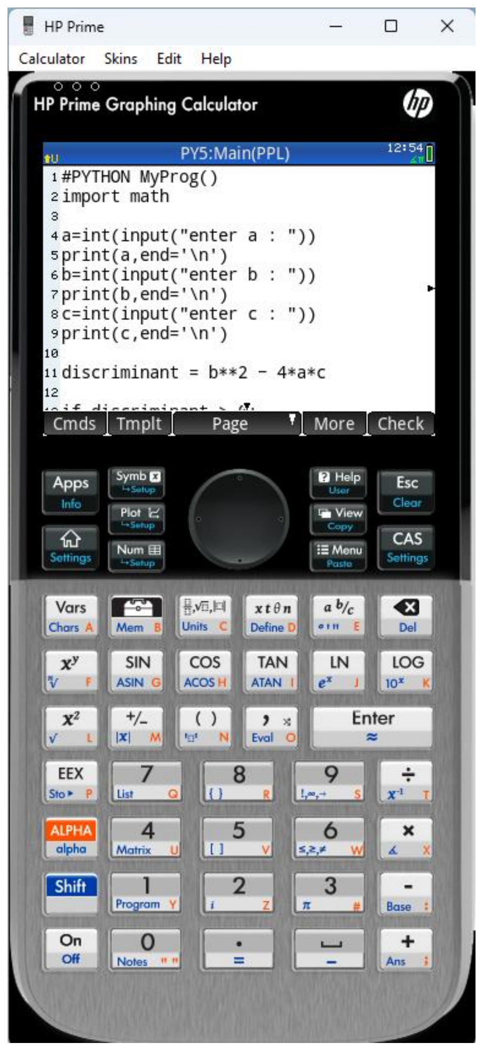


The highlighted code can then be cut and pasted into the program code editor in the HP-Prime Connectivity kit.

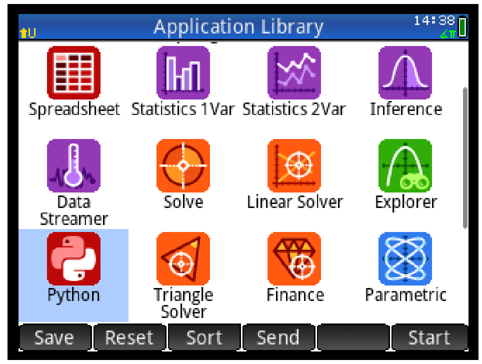


Then click on save all on the menu bar and this will be sent to the calculator where you can select and run the program.





4. Creating Python Applications on the Prime

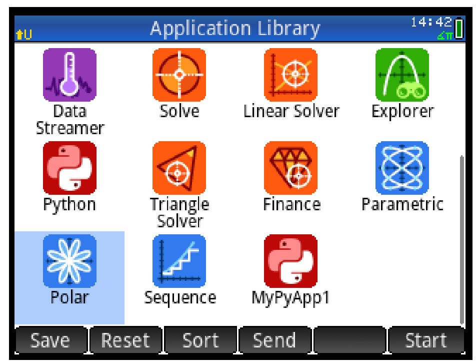


Creating a Python application is relatively straightforward but less tightly integrated with HPPL.

Step 1 – create an empty app, based on the Python app by selecting the Python app, then choosing Save at the bottom on the menu bar.

Step 2 – Give it a name and base it on the Python app. I have called mine MyPyApp1.

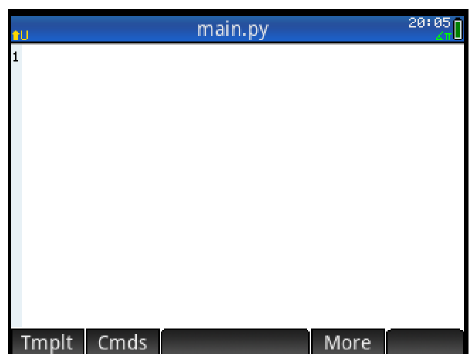
Step 3 – Press OK twice, a new application called MyPyApp1 will be created and appear on the apps screen.



You can rename the new app to whatever you wish, you can also give it a custom icon and add files to the application.

To see the structure and where to put the files you need to load the Connectivity Kit. If it doesn't show the new app, then highlight the calculator and right click and select Refresh which will synch the connectivity kit with the calculator.

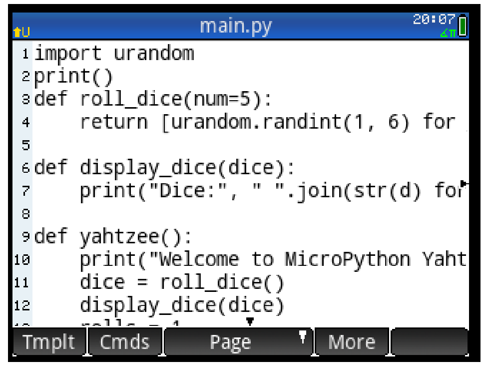
OK, let's create a Python app that recreates a simple one round Yahtzee game.



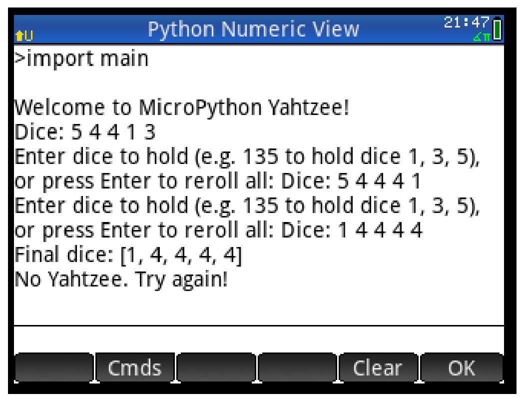
With the new Python App highlighted select the Symb key This should bring up a blank main.py with line number 1 showing on the left. Now create the following code in VS Code and copy and paste it into the main.py screen editor.

```
import urandom
print()
def roll_dice(num=5):
    return [urandom.randint(1, 6) for _ in range(num)]

def display_dice(dice):
    print("Dice:", " ".join(str(d) for d in dice))

def yahtzee():
    print("Welcome to MicroPython Yahtzee!")
    dice = roll_dice()
    display_dice(dice)
    rolls = 1
    while rolls < 3:
        hold = input("Enter dice to hold (e.g. 135 to hold dice 1, 3, 5), or press Enter to reroll all: ")
```

```
        if hold.strip() == "":
            dice = roll_dice()
        else:
            hold_indices = [int(i)-1 for i in hold if
i.isdigit() and 1 <= int(i) <= 5]
            new_dice = []
            for i in range(5):
                if i in hold_indices:
                    new_dice.append(dice[i])
                else:
                    new_dice.append(urandom.randint(1,
6))
            dice = new_dice
        display_dice(dice)
        rolls += 1
    # Simple scoring: check for Yahtzee
    if dice.count(dice[0]) == 5:
        print("YAHTZEE! All dice are", dice[0])
    else:
        print("Final dice:", dice)
        print("No Yahtzee. Try again!")

yahtzee()
```



Now the first time after you edit it when you click on the app you should get the following prompt. Press or click on OK to import main.py. It will then import and run the program you should see the following output.



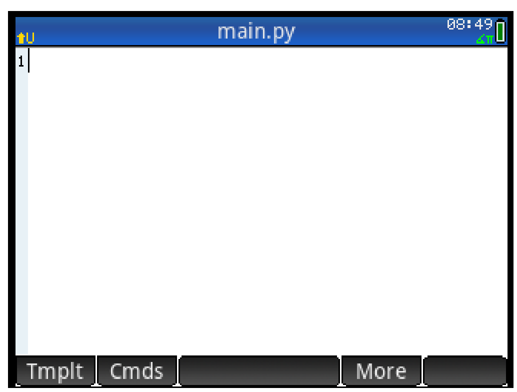
5. Python application windows

When you select a python application the functions of the Synb and Num keys change.

The Symb key brings up the code window by default showing main.py with line numbers down the left hand side.

This is where you manually enter micropython code for the application. If you enter it or change it the application will ask if you want to re-import main.py the next time it is run.

The Num key brings up the console or terminal window, this is where you see the results of the program as in the Yahtzee example above in Python Numeric View.



6. Operators

MicroPython can perform various mathematical operations using primitive and logical operations.

Type	Operator	Name	Example
Arithmetic	+	Addition	variable + 1
	-	Subtraction	variable - 1
	*	Multiplication	variable * 4
	/	Division	variable / 4
	%	Modulo division	variable % 4
Comparison	==	Equals	expression1 == expression2
	!=	Not equal	expression1 != expression2
	<	Less than	expression1 < expression2
	>	Greater than	expression1 > expression2
	<=	Less than or equals	expression1 <= expression2
	>=	Greater than or equals	expression1 >= expression2
Logical	&	bitwise and	variable1 & variable2
	|	bitwise or	variable1 | variable2
	^	bitwise exclusive or	variable1 ^ variable2
	~	bitwise complement	~variable1
	and	logical and	variable1 and variable2
	or	logical or	variable1 or variable2

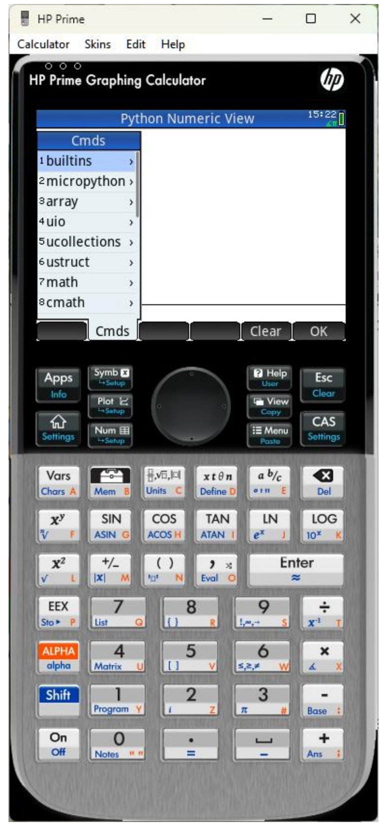
Supported operations

7. Python libraries

There are three types of libraries in MicroPython:

1. derived from a standard Python library (built-in libraries)
2. specific MicroPython libraries
3. specific libraries to assist with hardware functionality i.e. the HP-Prime.

The list can be found on the calculator by starting the Python app and choosing the CMDS item in the bottom menu.



The Prime implements a subset of micropython whose full documentation is found at the Micropython site. Be aware that there are more routines documented there than exist in the Prime. For easier searching while writing code, the list below might be convenient.

Standard Python libraries in MicroPython[4]

Library name	Description	Functions			
array	operations on arrays	asc char euler	gcd iegcd ifactor	isprime lcm nextprime	nprimes prevprime
cmath	provides math functions for complex numbers	cos e exp log	log10 phase pi polar	rect sin sqrt	
gc	garbage collector	collect disable enable isenabled	mem_alloc mem_free threshold		
math	provides basic math operations for floating-point numbers	acos acosh asin asinh atan atan2 atanh ceil copysign cos	cosh degrees e erf erfc exp expm1 fabs floor fmod	frexp gamma isfinite isinf isnan ldexp lgamma log log10 log2	modf pi pow radians sin sinh sqrt tan tanh trunc

sys	system-level functions; provides access to variables used by the interpreter	argv byteorder exc_info exit	implementation maxsize modules path platform	print_exception stderr stdin stdout version version_info	
ucollections	operations for collections and container types that hold various objects	deque append popleft namedtuple	OrderedDict clear copy fromkeys get items keys pop popitem setdefault update values		
uerrno	provides access to error codes	errorcode EACCES EADDRINUSE EAGAIN EALREADY EBADF	ECONNABORTED	ECONNREFUSED ECONNRESET EEXIST	EHOSTUNREACH EINPROGRESS EINVAL EIO EISDIR ENOBUFS ENODEV ENOENT ENOMEM ENOTCONN EOPNOTSUPP EPERM ETIMEDOUT
uhashlib	operations for binary hash algorithms	sha256 digest update			
uio	operations for handling input/output streams	open BytesIO close flush getvalue read readinto readline readlines seek write	FileIO close fileno flush read readinto readline readlines seek tell write	StringIO close flush getvalue read readinto readline readlines seek write	TextIOWrapper close fileno flush read readinto readline readlines seek tell write

ure	implements regular expression matching operations	compile match search DEBUG			
ustruct	performs conversions to Python objects by packing and unpacking primitive data types	calcsize pack pack_into	unpack unpack_from		
utimeq	provides time and date function, including measuring time intervals and implementing delays	utimeq peektime pop push			
Urandom	Random numbers functions	choice getrandbits randint random randrange seed uniform			

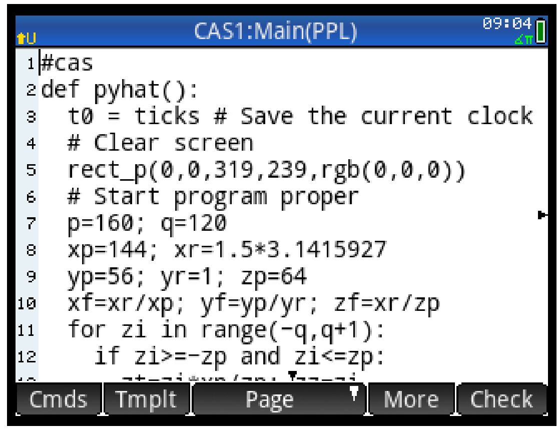
HP-Prime MicroPython-specific libraries[4]

Library name	Description				
micropython	access and control of MicroPython internals like the library heapq	const heap_lock heap_unlock kbd_intr mem_info opt_level pystack_use qstr_info stack_use			
Builtins	The builtins module contains all the basic Python functions and types	abs all any bin bool bytearray append extend bytes center count decode endswith find format	divmod Ellipsis enumerate eval exec filter float frozenset copy update values dir difference intersection	NotImplemented object oct open ord pow print property deleter getter setter range repr reversed round	join lower lstrip partition replace rfind rindex rpartition rsplit rstrip split splitlines startswith strip upper

		index isalpha isdigit islower isspace isupper join lower lstrip parition replace rfind rindex rpartition rsplit rstrip split splitlines startswith strip upper callable chr classmethod compile complex delattr dict clear copy fromkeys get items keys pop popitem setdefault	isdisjoint issubset issuperset symmetric_difference union getattr globals hasattr hash help hex id input int from_bytes to_bytes isinstance issubclass iter len list append clear copy count extend index insert pop remove reverse sort locals map max memoryview min next	set add clear copy difference difference_update discard intersection intersection_update isdisjoint issubset issuperset pop remove symmetric_difference symmetric_difference_update union update setattr slice sorted str center count encode endswith find format index isalpha isdigit islower isspace isupper	staticmethod sum super tuple count index type zip ArithmeticError AssertionError AttributeError BaseException EOFError Exception GeneratorExit ImportError IndentationError IndexError KeyboardInterrupt KeyError LookupError MemoryError NameError NotImplementedError OSError OverflowError RuntimeError StopAsyncIteration StopIteration SyntaxError SystemError TypeError UnicodeError ValueError ZeroDivisionError
hpprime	Is the main interface between Python and the HP Prime subsystem - It provides mainly drawing primitives for fast graphics and the eval function which allows Python to interact with the rest of the Prime system.	arc arc_c blit blit_c circle circle_c dimgrob dimgrob_c eval	fillrect fillrect_c get_cartesian grob grob_c grobh grobh_c grobw	grobw_c keyboard line line_c mouse pixon pixon_c	rect rect_c set_cartesian strblit strblit_c textout textout_c

Graphic	Pixel and colour drawing functions.	cyan magenta yellow black white red green blue	draw_filled_circle draw_filled_polygon draw_line draw_pixel	draw_polygon draw_rectangle draw_string get_pixel set_pixel show show_screen	clear_screen draw_arc draw_circle draw_filled_arc
CAS	Evaluate CAS expressions from within prime.	caseval eval_expr get_key xcas			
Arit	Arithmetic functions - Prime numbers	asc char euler gcd iegcd ifactor	isprime lcm nextprime nprimes prevprime		
Linalg	Linear Algebra functions	abs add apply arange conj cross det dot egv eig eigenvects eye fft horner identity idn ifft im imag	inv linspace matrix mul ones pcoeff peval pi proot rand ranm	ranv re real rref shape size solve sub transpose zeros	
matplotl	Chart plotting functions	arrow axis bar barplot boxplot	boxwhisker clf grid hist histogram		

8. Python from CAS



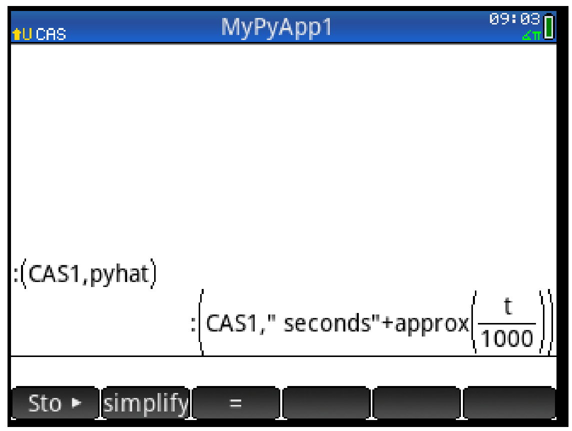
You can also use Python syntax in a CAS program - first create and save the code exactly as is, in a program. Delete any template code (EXPORT, BEGIN, END) lines.

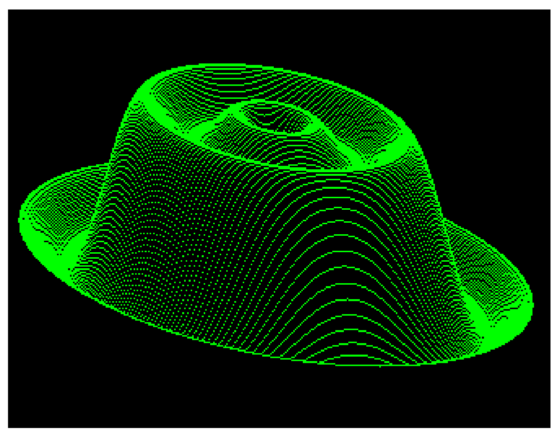
Run the function, by calling <name of program>:<name of function>()

For example, if code is saved as CAS1,

in cas screen, type test01:pyhat()

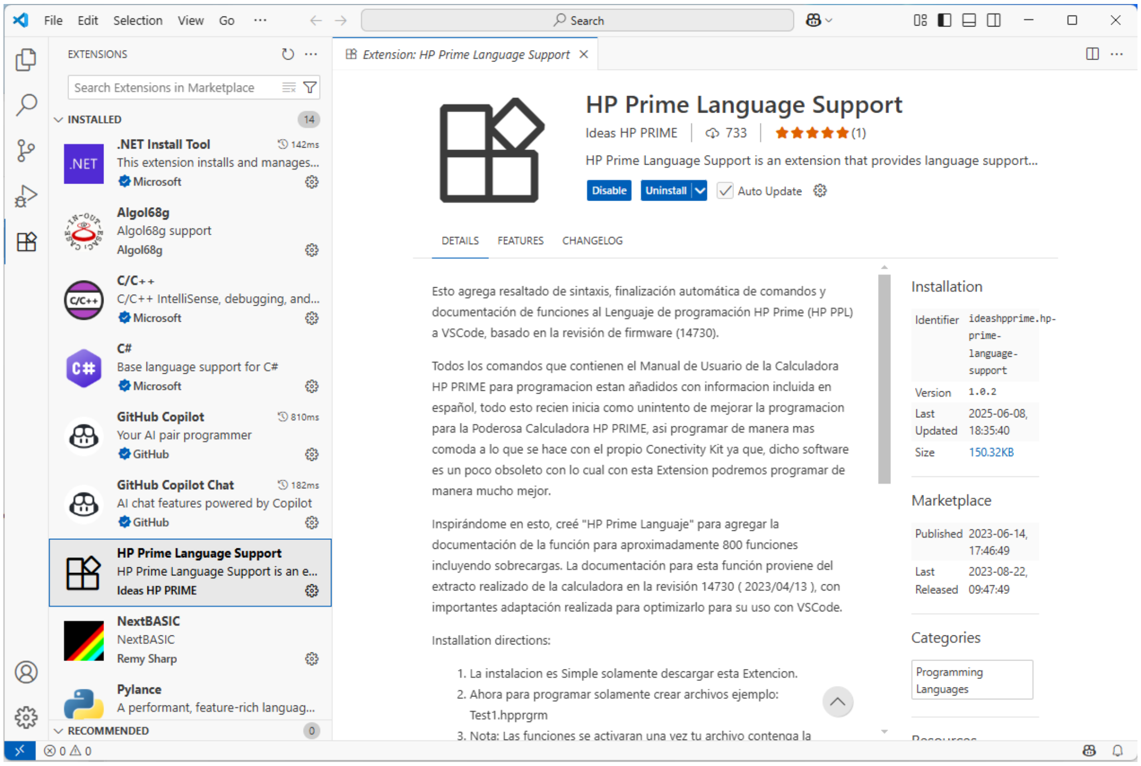
```
#cas
def pyhat():
  t0 = ticks # Save the current clock count for timing program
  # Clear screen
  rect_p(0,0,319,239,rgb(0,0,0))
  # Start program proper
  p=160; q=120
  xp=144; xr=1.5*3.1415927
  yp=56; yr=1; zp=64
  xf=xr/xp; yf=yp/yr; zf=xr/zp
  for zi in range(-q,q+1):
    if zi>=-zp and zi<=zp:
      zt=zi*xp/zp; zz=zi
      xl=int(.5+sqrt(xp*xp-zt*zt))
      # Draw one cross-section of figure
      for xi in range(-xl,xl+1):
        xt=sqrt(xi*xi+zt*zt)*xf; xx=xi
        yy=(sin(xt)+.4*sin(3*xt))*yf
        x1=xx+zz+p
        y1=yy-zz+q
        pixon_p(x1,230-y1,rgb(0,255,0))
        if y1!=0:
          Line_p(x1,230-y1+1,x1,230) # Erase points below current point
  t = ticks-t0
  # Wait for key and print elapsed time
  wait
  print(approx(t/1000)+" seconds")
#end
```



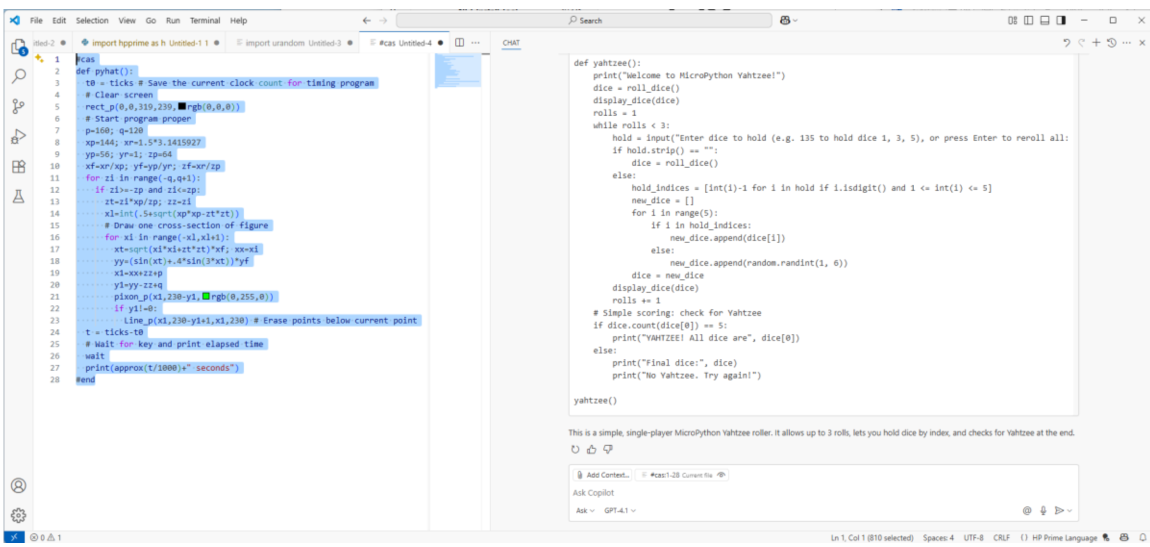


9. Using VS Code to wite PPL and Python

Visual studio code is a programmer's editor with many language extensions. One extension is the HP Prime Language Support extension.



Once installed it understands HP Prime PPL and embedded Python syntax.This includes colour coding keywords and correctly indenting the code.



It also includes the ability to use co-pilot to help fix and generate code. Once written in VS Code the code can be copied and then pasted straight into either the virtual calculator program window or into the program in the Connectivity kit.

10. Examples

Example 1. Calling Prime functions from Python

```
#PYTHON name
from math import *
from hpprime import *
t = eval("ticks")
loops = 30000
for i in range(loops):
    r0 = 10
    while True:
        x = r0
        x += 1
        x -= 4.567E-4
        x += 70
        x -= 69
        x *= 7
        x /= 11
        r0 -= 1
        if r0 <= 0:
            break
    x = log(x)
    x = sin(x)
    x = sqrt(x)
    x = sqrt(x)
print(x)
t = (eval("ticks")-t)/1000
print("Loops:", loops)
print("Time: {0:.3f} seconds".format(t))
print("Index: {0:.2f}".format(34/t*loops))
#end

EXPORT calcperf()
BEGIN
 print();
 PYTHON(name);
END;
```

11. Links

FORTH written in Python GitHub - diemheych/PrimeFORTH: A simple version of FORTH written in Python for the HP Prime calculator

Classic adventure – Collosal caves written in Python
https://udel.edu/~mm/hp/adventure/

Hunt the Wumpus written for the HP Prime

HP Prime plays Hunt the Wumpus in Python