2 What is a Program?

In short a program is a computer application that you create, using a language of commands that the computer *interprets.* A program is also a document meant to be readable by both the interpreter, i.e. python, and by humans. The goal of this book is to get you to be able to write a computer program that can solve mathematical problems. A python program is a computer document with a sequence of instructions in the python language, in our case, that the computer executes. Roughly, you specify the instructions, concluding with those to report out the answers you seek, then you run the program. This cycle of writing, running, then seeing the results over and over it what its all about. The easier and quicker the cycle is, the quicker your progress. BASIC was notable for the features it introduced to improve **interaction**. This book introduces python's interactive environment incrementally, giving you more and more "tricks" that skilled programmer's prize.

The first chapter explains how to specify mathematical expressions, and subsequent chapters each introduce another language instruction or feature—there are about a dozen you need to know, out of the hundreds that there are. Each chapter starts with a concise section that introduces the instructions in python sufficient to duplicate what BASIC can do. After introducing the "BASICs" each chapter has **Notes** on how python extends BASIC's functionality, and clues on how programming language features have evolved since BASIC's early days, to be read at your leisure, if your are curious.

3 A BASIC Primer, in python

3.1 Expressions, Functions, Numbers and Variables

When you are at the >>> prompt, you can *evaluate* expressions to find their value. Type the expression followed by the "enter" key

and its value is printed on the next line:

Example 5: Evaluating an expression in the Interpreter.

```
>>> (3 * 60 + 18) / 3600
0.055
```

Mathematical expressions in python language resemble textbook algebraic expressions, for example `38 / 2`, or `2 * ( -0.05 + 17)`. The computer reduces a valid expression to a single number, working from left to right, and following conventional rules of applying multiplication and division before addition and subtraction, and evaluating sub-expressions in parenthesis before the expressions that contain them. In expressions python ignores spaces, however humans like them for readability.

The four basic *binary operators*, $+$, $-$, $*$, and $/$ combine the values immediately before and after them. Multiplication requires $*$ for times. Division must be written out using $/$ rather than "stacking" numerator and denominator with a line between them. This requires adding parenthesis to express the grouping implicit in the numerator and denominator of a fraction. It's also handy to know the binary operators $//$, and $\%$, for integer division and modulo. Integer division returns just the integer part of a quotient, and modulo returns the remainder. Its always true that $x/y = x//y + (x\%y)/y$. ▸

The python operators are largely identical to BASIC's, with the notable exception of it's exponentiation operator, for which the python function `pow(,)` can be used.

Numbers with a fractional part are written with " `.` " a decimal point. Commas to separate thousands and millions are not allowed, neither are embedded spaces. Negative numbers are preceded by $-$, just as positive numbers can be preceded by $+$, which is entirely unnecessary. Preceding signs can be separated from digits by spaces since they are actually *unary* operators, not strictly part of the number.

For values too large or small ▸ to be written with a decimal point there's a convention for writing exponential notation in base 10 using " `E`" notation, in the form of the mantissa, followed by `E` or `e`, followed by the exponent. This is one place where capitalization

Numbers can have about 16 significsant digits.

doesn't matter. This is just how scientific notation is written; e.g 3.9×10^9 can be written either as `3.98E9` or `3.98e9`.

This is a more complicated expression demonstrates these points:

$$\frac{\frac{1}{2.34\times10^{-5}}}{1-7(0.1+1.4142135623730951)}$$

is written as an expression:

Example 6: A complicated expression written in python.

```
(1 / ( 2.34E-5))/(1 - 7 * ( 0.1 + 1.4142135623730951)).
```

If you type this to the prompt it evaluates to `-5211.911595216185`.

Expressions can also include functions provided by python, and program-defined variables, as we explain.

3.1.1 Modules Containing Functions

Functions in an expression are written as the function name, preceded by their *module* name, e.g. `math.sqrt`, and followed by the argument or arguments enclosed in parentheses, just like you'd write them in mathematical notation. So the sine of zero is written `math.sin(0)`. Since the parentheses' purpose is to let the function know what it's arguments are, parentheses that delimit a function's argument list are different than grouping parentheses —still the function together with its argument list implicitly works as a group. An expression in the place of a function argument is evaluated as passed to the function before the function is evaluated. If there is more than one argument in the argument list, they are separated by commas. The interpreter *returns* the value of the function when it evaluates it, so, to demonstrate,

Example 7: Calling math package functions.

```
>>> import math
>>> 1 + math.sin( math.pi / 2)
2.0
```

python has numerous functions, so to keep track of them they are organized into sets called *modules.*▸ Before a function can be called, it's module must be *imported.* When the function is called the name of the module precedes the function name, separated from it by a period. For the set we are interested in you'd load the `math` module with the statement `>>> import math` before calling any functions.

More generally there are numerous *packages* that extend python's functionality. Precisely a package is a set of modules, but we will use the terms interchangeably, when there's no confusion.

Many common functions, such as $sin()$, $log()$, et cetera▸ can be called by name, prefaced by " `math.`". Exponentiation is also written as the function " `math.pow(,)`". So the previous Example 6 could also be written:

For a list of all of them type `help(math)`. Remember you will have to have already typed `import math` during the session.

Example 8: An expression containing math package functions.

```
import math
(math.pow(2.34E-5, -1) )/( 1 + 7 * (0.1 - math.sqrt(2))).
```

It will return the same value.

3.2 The python Interpreter

By typing numeric expressions on a single line to the prompt, the interpreter can be used as a calculator. A few other features of the interpreter make this more useful. An expression can extend over several lines if begun with an open parenthesis. While you are writing an multi-line expression the prompt will change to .... The interpreter will not complete the expression until a matching close parenthesis ends a line. Another useful feature is that the interpreter keeps in memory a list of your previous expressions as *history*. Typing the up-arrow key retrieves the previous typed line, which can b e edited, then evaluated by hitting enter. Other previous lines can be recalled in sequence using the up-arrow and down-arrow keys repeatedly. Similarly the value from the last expression evaluated can be retrieved by the symbol "_". Using these tricks one can speed up repetitive manual computations.

Example 9: Using the previous computed value in the Interpreter.