Turtle Python 1 – Getting Started

1. Introducing the Turtle System

The *Turtle System* (or just *Turtle* for short) is a programming environment designed for ease of use and learning, offering three programming languages:

- *Turtle Python* (the default)
- *Turtle BASIC*
- *Turtle Pascal*

These are simplified versions of standard languages, which makes it easy to get started, more manageable for students (and teachers), and also enables *Turtle* to provide lots of help for beginners. For example, the error messages can be far more precise and straightforward than in an “industrial strength” system – enabling the vast majority of elementary errors to be sorted by the student without further help. Another virtue of this simplicity is to enable straightforward comparison of the different languages (for example, by swapping between the example programs and seeing how they change between the languages). This helps to demonstrate that programming in *any* language of this general type is essentially similar: what matters most is not the detailed “syntax” of the specific language, but rather the *logic* of the *algorithms* (i.e. the computational recipes) that the language expresses. So it really doesn’t matter what programming language you start with: once you have developed the skill of *thinking algorithmically*, that skill can easily be transferred to another language, and you can use the same *Turtle System* to start learning it with minimal difficulty!

2. Downloading and Running the System

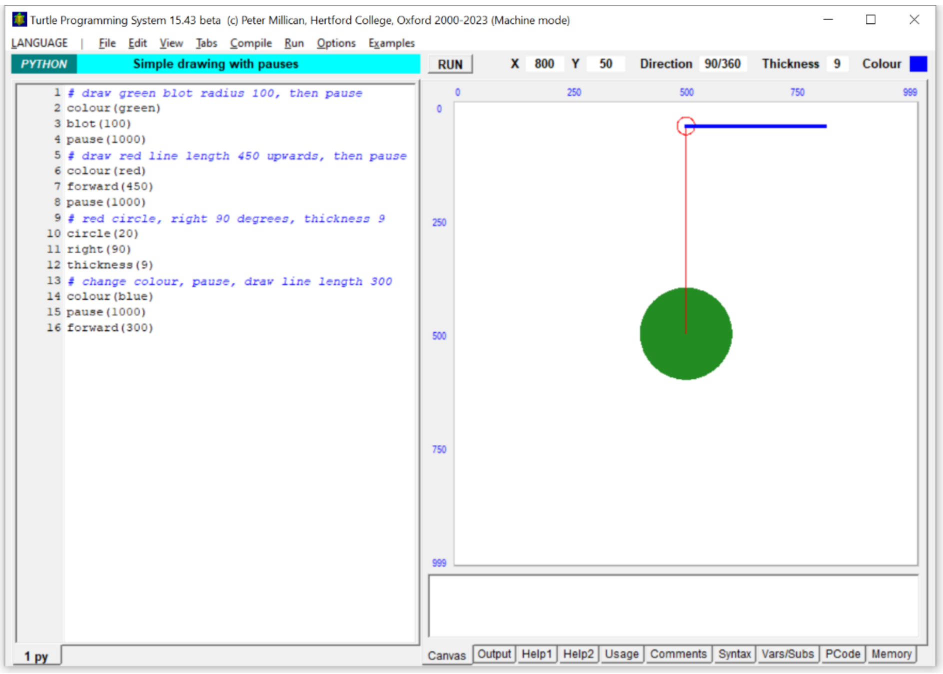
Turtle can be downloaded free of charge from www.turtle.ox.ac.uk, and the main system consists of one executable program – named *TurtleSystem_15.exe* – which can be run from any folder on a PC (a version for Mac is planned for 2024). No further installation is necessary, and the system will never make any secret changes to your computer setup.

After you have moved the executable program to an appropriate directory (e.g. maybe “C:\Turtle”), you can run it simply by double-clicking on it within *Windows Explorer*. At this point, if you have not run it before, you might get a warning saying “Windows protected your PC”, indicating that your system has identified a potential threat in the form of an unknown program. Needless to say, we have no malicious software on www.turtle.ox.ac.uk, so you can safely click on the “More info” link and on the button that should appear saying “Run anyway”. On running the program first time, you are also likely to see a dialog box headed “Setting Up Your Personal Turtle Directory”. If you have no wish to experiment with file processing programs, it’s fine to click on “Continue without Turtle Directory”. Otherwise, select one of the three offered options and click on “Attempt to Set Up Selected Directory” – if, for example, you select the third option, then you should find that your “Turtle” directory now has a subdirectory called “Turtle Files”. (Note that as a security measure and to avoid accidental modification of other computer files, the *Turtle System* will not allow your programs to process files that are not within a directory named “Turtle Files”.)

When the system starts up, you should see at the left that it is running *“PYTHON”*, and that there are five menus to the right of this along the top, “File”, “Edit”, “View”, “Run”, “Options” and “Examples”. If you now click on the “View” menu and select “Simple Mode” (near the bottom), you will find that the “Options” menu disappears and all of the menus (except for “Examples”) become shorter. The system supports four “Modes”, to enable non-expert users to be shielded from the more sophisticated facilities that they are unlikely to use and might find distracting or confusing.

3. Starting with “Turtle Graphics”

Staying in Simple Mode, click on the “Examples” menu and select the very first example program from the first sub-menu – it is called “Simple drawing with pauses”. Now click on the “RUN” button (just below and a bit to the right of the menus). You should then see something like this:


Turtle Programming System 15.43 beta (c) Peter Millican, Hertford College, Oxford 2000-2023 (Machine mode)
LANGUAGE | File Edit View Tabs Compile Run Options Examples
PYTHON
Simple drawing with pauses
RUN
X 800 Y 50 Direction 90/360 Thickness 9 Colour
0 250 500 750 999
0 250 500 750 999
1 py
Canvas Output Help1 Help2 Usage Comments Syntax Vars/Subs PCode Memory

```
# draw green blot radius 100, then pause
colour(green)
blot(100)
pause(1000)
# draw red line length 450 upwards, then pause
colour(red)
forward(450)
pause(1000)
# red circle, right 90 degrees, thickness 9
circle(20)
right(90)
thickness(9)
# change colour, pause, draw line length 300
colour(blue)
pause(1000)
forward(300)
```

The name of the *Turtle System* is derived from Seymour Papert’s invention of “turtle graphics” as a brilliant way to introduce programming to children (of all ages!). Although originally associated with his programming language *LOGO*, this has proved so successful and popular that most standard languages now have at least one turtle graphics module available for learners (and often there are numerous versions, produced all over the world). The idea is to imagine an invisible *Turtle* (though children’s systems will often provide a visible image) who starts off pointing “north” in the middle of a drawing “Canvas”, and is given drawing instructions by the written program. In this case, when “RUN” is clicked, the *Turtle* is given the following instructions:

```
colour(green)
blot(100)
pause(1000)
colour(red)
forward(450)
pause(1000)
circle(20)
right(90)
thickness(9)
colour(blue)
pause(1000)
forward(300)
```

(The first line of the program, “# draw green blot radius 100, then pause”, is a *comment* rather than an instruction – it’s included to make the program easier to understand. If you want to delete all of the comments, select “Remove any comments” from the “Edit” menu, and note that if you then want to reverse this, use “Restore previous version” – at the bottom of that menu – rather than “Undo”.)

You will see that the program has three `pause(1000)` instructions – pausing for 1000 milliseconds (i.e. one second) each time – so when it runs you can see its progress in steps. By the first pause, the *Turtle* has drawn a green blot (i.e. a filled green circle) of radius 100 in its original position at the centre of the Canvas. By the second pause, it has changed its pen colour to red and drawn a thin red line northwards, nearly reaching the top of the Canvas (1000×1000 units by default, so going north 450 units leaves the *Turtle* 50 units from the top). Then it turns right by 90 degrees, changes its pen thickness to 9 (from the default of 2) and its pen colour to blue, pauses a third time, and finally draws a thick blue line 300 units long (pointing now towards the east, after its 90 degree turn). Note that between the second and third pauses, the Canvas itself doesn't change at all, but you can see the changes in the *Turtle*'s direction, pen thickness and colour in the information boxes above the Canvas. The values in these boxes can be used in your programs, using the names:

turtx	the current x-coordinate of the *Turtle*
turty	the current y-coordinate of the *Turtle*
turtd	the current direction of the *Turtle* (expressed in "angle units")
turta	the number of "angle units" in a circle (by default, 360)
turtt	the current thickness of the *Turtle*'s pen
turtc	the current colour of the *Turtle*'s pen (expressed as an "RGB" colour code)

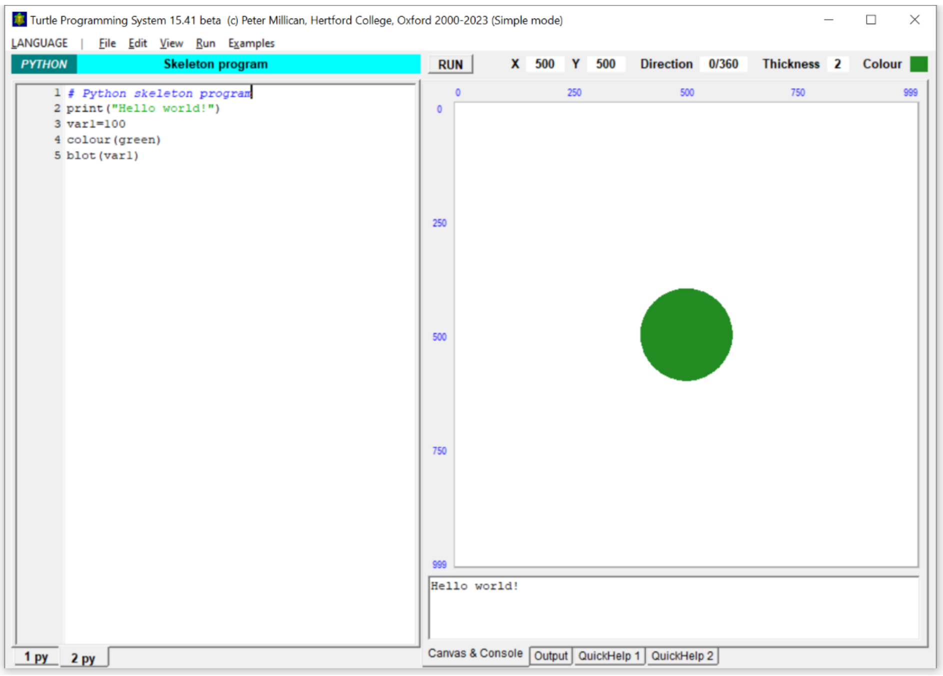
We won't be using any of these right now, but it's helpful to remember that the current values of these *Turtle attributes* will affect, for example, where lines are drawn and in what colour etc. Thus the effect of the instruction `thickness(9)`, for example, is not to actually draw anything with that thickness, but just to change the *Turtle's* current pen thickness attribute, *turtt*, to 9.

4. Canvas, Console, and Output

Run that first program a few times until you are sure that you understand what it's doing. Another very simple program can be loaded by choosing "Skeleton program" from the "File" menu; this has just four instructions:

```
print("Hello world!")
var1=100
colour(green)
blot(var1)
```

When you now click "RUN" you should see a familiar green blot in the centre of the Canvas, but also the text "Hello world!" in the Console beneath the Canvas:



Turtle Programming System 15.41 beta (c) Peter Millican, Hertford College, Oxford 2000-2023 (Simple mode)

```
# Python skeleton program
print("Hello world!")
var1=100
colour(green)
blot(var1)
```

X 500 Y 500 Direction 0/360 Thickness 2 Colour 🟩

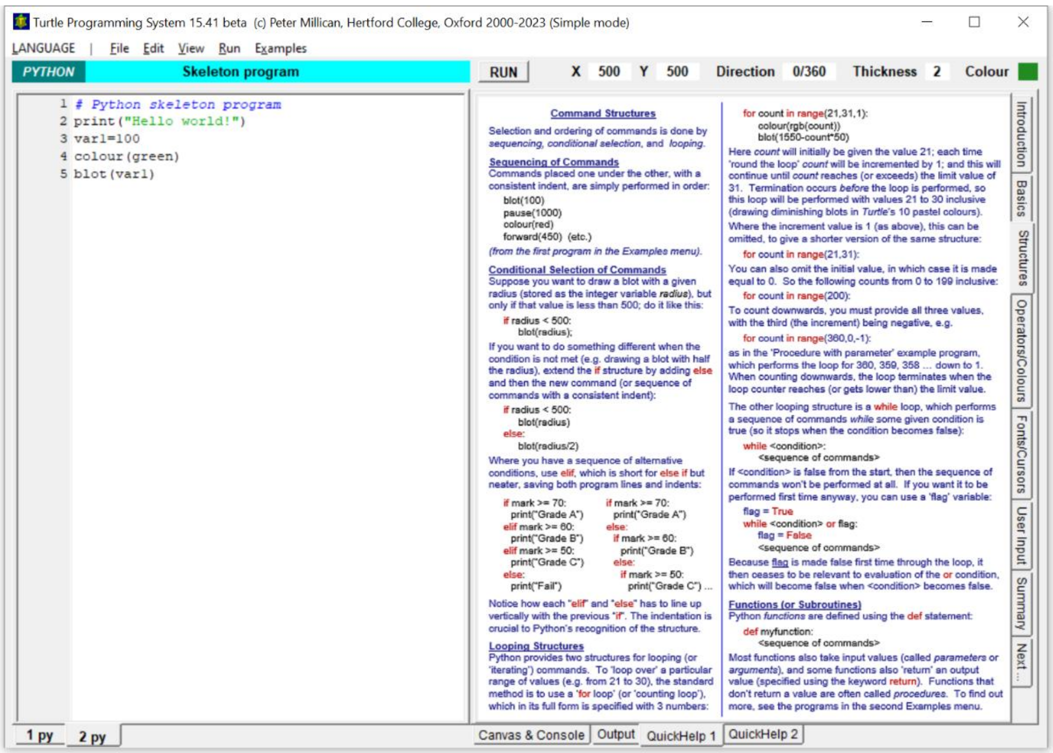
```
Hello world!
```

Any text written to the Console is also standardly written also to the “Output” tab, which can be selected at the bottom right of the software window. Try clicking on that, then immediately on “RUN” again. You will see that running a program standardly brings you back to the “Canvas & Console” tab (though this behaviour can be changed, in more advanced system Modes, from the “Run” menu).

The point of the “Skeleton program” is to give a very simple example of a program, including a comment, a print instruction, the setting of a numeric variable (here called “var1”, and set to 100) and of the *Turtle*’s colour, and the drawing of a blot of the given size. To appreciate the value of this, try selecting “Turtle Pascal” from the “LANGUAGE” menu: you are instantly into Pascal programming – with a comment, a print instruction, a defined and assigned variable etc. – without having to first learn for yourself about Pascal program structure. Now go back to “Turtle Python”.

5. The Help Panels

The two other tabs visible at the bottom right of the software window in Simple Mode are labelled “QuickHelp 1” and “QuickHelp 2”. The first of these contains eight information panels, outlining in a very abbreviated way the main points of programming in Turtle Python. This material is not primarily designed for *learning*, but rather as a useful *prompt* or *reminder* that can save a lot of time if you forget details while writing a program. (These panels are also very helpful if, say, you want to quickly pick up the fundamentals of programming in another language, such as Turtle Pascal.)



Turtle Programming System 15.41 beta (c) Peter Millican, Hertford College, Oxford 2000-2023 (Simple mode)
LANGUAGE | File Edit View Run Examples
PYTHON
Skeleton program
RUN
X 500 Y 500 Direction 0/360 Thickness 2 Colour
Introduction
Basics
Structures
Operators/Colours
Fonts/Cursors
User Input
Summary
Next ...
1 py
2 py
Canvas & Console
Output
QuickHelp 1
QuickHelp 2

```
# Python skeleton program
print("Hello world!")
var1=100
colour(green)
blot(var1)
```

Command Structures

Selection and ordering of commands is done by *sequencing, conditional selection*, and *looping*.

Sequencing of Commands
Commands placed one under the other, with a consistent indent, are simply performed in order:

```
blot(100)
pause(1000)
colour(red)
forward(450)  (etc.)
```

(from the first program in the Examples menu).

Conditional Selection of Commands
Suppose you want to draw a blot with a given radius (stored as the integer variable *radius*), but only if that value is less than 500; do it like this:

```
if radius < 500:
  blot(radius);
```

If you want to do something different when the condition is not met (e.g. drawing a blot with half the radius), extend the if structure by adding else and then the new command (or sequence of commands with a consistent indent):

```
if radius < 500:
  blot(radius)
else:
  blot(radius/2)
```

Where you have a sequence of alternative conditions, use elif, which is short for else if but neater, saving both program lines and indents:

```
if mark >= 70:           if mark >= 70:
  print("Grade A")         print("Grade A")
elif mark >= 60:         else:
  print("Grade B")         if mark >= 60:
elif mark >= 50:             print("Grade B")
  print("Grade C")         else:
else:                        if mark >= 50:
  print("Fail")                print("Grade C") ...
```

Notice how each "elif" and "else" has to line up vertically with the previous "if". The indentation is crucial to Python's recognition of the structure.

Looping Structures
Python provides two structures for looping (or 'iterating') commands. To 'loop over' a particular range of values (e.g. from 21 to 30), the standard method is to use a 'for loop' (or 'counting loop'), which in its full form is specified with 3 numbers:

```
for count in range(21,31,1):
  colour(rgb(count))
  blot(1550-count*50)
```

Here *count* will initially be given the value 21; each time 'round the loop' *count* will be incremented by 1; and this will continue until *count* reaches (or exceeds) the limit value of 31. Termination occurs *before* the loop is performed, so this loop will be performed with values 21 to 30 inclusive (drawing diminishing blots in *Turtle*'s 10 pastel colours).

Where the increment value is 1 (as above), this can be omitted, to give a shorter version of the same structure:

```
for count in range(21,31):
```

You can also omit the initial value, in which case it is made equal to 0. So the following counts from 0 to 199 inclusive:

```
for count in range(200):
```

To count downwards, you must provide all three values, with the third (the increment) being negative, e.g.

```
for count in range(360,0,-1):
```

as in the 'Procedure with parameter' example program, which performs the loop for 360, 359, 358 ... down to 1. When counting downwards, the loop terminates when the loop counter reaches (or gets lower than) the limit value.

The other looping structure is a while loop, which performs a sequence of commands *while* some given condition is true (so it stops when the condition becomes false):

```
while <condition>:
  <sequence of commands>
```

If <condition> is false from the start, then the sequence of commands won't be performed at all. If you want it to be performed first time anyway, you can use a 'flag' variable:

```
flag = True
while <condition> or flag:
  flag = False
  <sequence of commands>
```

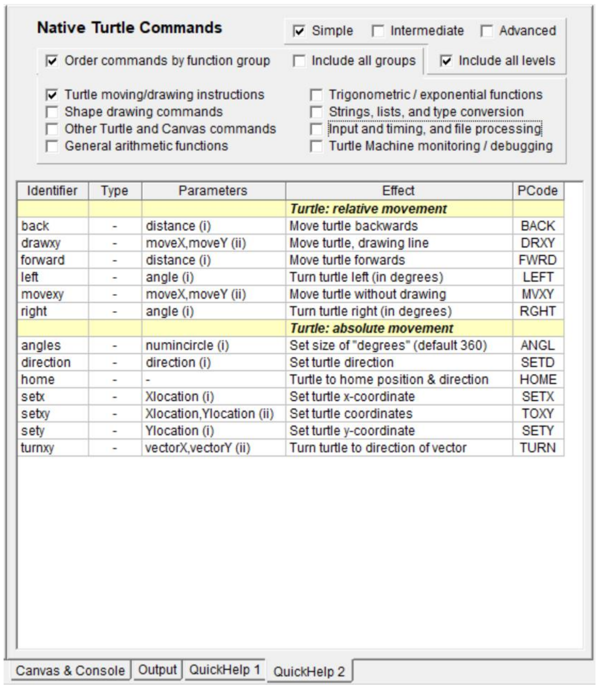
Because flag is made false first time through the loop, it then ceases to be relevant to evaluation of the or condition, which will become false when <condition> becomes false.

Functions (or Subroutines)
Python *functions* are defined using the def statement:

```
def myfunction:
  <sequence of commands>
```

Most functions also take input values (called *parameters* or *arguments*), and some functions also 'return' an output value (specified using the keyword return). Functions that don't return a value are often called *procedures*. To find out more, see the programs in the second Examples menu.

The “QuickHelp 2” tab is even more helpful, giving a summary of Turtle Python commands which will often be sufficient for you to use them effectively in your programs. If you are in Simple Mode, then relatively few commands will be shown initially, but you can if you wish display all of the commands available, either alphabetically or (the default) arranged by function groups, starting with *Turtle* movement commands, then shape drawing commands, and so on. The commands are also classified into “Simple”, “Intermediate”, and “Advanced”, but if you select both “Include all levels” and “Include all groups”, you will see the entire range. More useful for advanced programmers is to “Include all levels” but to *deselect* “Include all groups” – after which the commands will all disappear from the table – and now select the particular function group that you are interested in, for example “Turtle moving/drawing instructions” or “Input and timing, and file processing”, as in the following images:

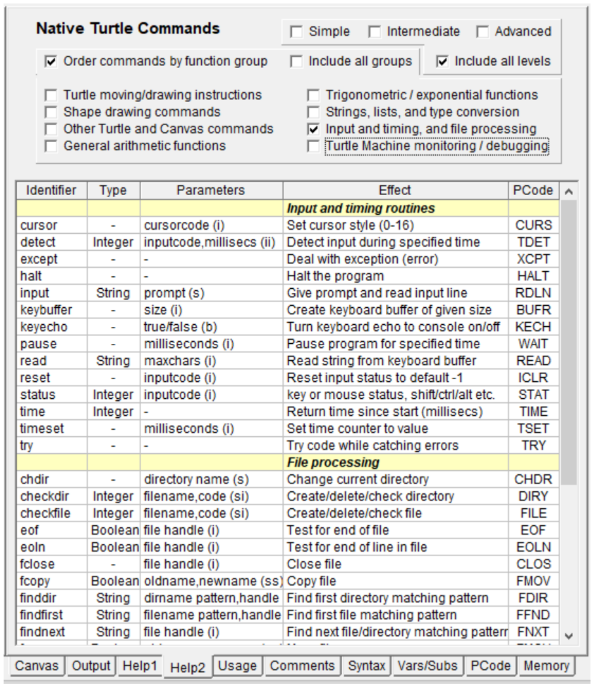

Native Turtle Commands

[✓ Simple] Intermediate Advanced

[✓ Order commands by function group] Include all groups [✓ Include all levels]

[✓ Turtle moving/drawing instructions] Shape drawing commands Other Turtle and Canvas commands General arithmetic functions Trigonometric / exponential functions Strings, lists, and type conversion Input and timing, and file processing Turtle Machine monitoring / debugging

Identifier	Type	Parameters	Effect	PCode
			Turtle: relative movement	
back	-	distance (i)	Move turtle backwards	BACK
drawxy	-	moveX,moveY (ii)	Move turtle, drawing line	DRXY
forward	-	distance (i)	Move turtle forwards	FWRD
left	-	angle (i)	Turn turtle left (in degrees)	LEFT
movexy	-	moveX,moveY (ii)	Move turtle without drawing	MVXY
right	-	angle (i)	Turn turtle right (in degrees)	RGHT
			Turtle: absolute movement	
angles	-	numincircle (i)	Set size of "degrees" (default 360)	ANGL
direction	-	direction (i)	Set turtle direction	SETD
home	-	-	Turtle to home position & direction	HOME
setx	-	Xlocation (i)	Set turtle x-coordinate	SETX
setxy	-	Xlocation,Ylocation (ii)	Set turtle coordinates	TOXY
sety	-	Ylocation (i)	Set turtle y-coordinate	SETY
turnxy	-	vectorX,vectorY (ii)	Turn turtle to direction of vector	TURN


Native Turtle Commands

Simple Intermediate Advanced

[✓ Order commands by function group] Include all groups [✓ Include all levels]

Turtle moving/drawing instructions Shape drawing commands Other Turtle and Canvas commands General arithmetic functions Trigonometric / exponential functions Strings, lists, and type conversion [✓ Input and timing, and file processing] Turtle Machine monitoring / debugging

Identifier	Type	Parameters	Effect	PCode
			Input and timing routines	
cursor	-	cursorcode (i)	Set cursor style (0-16)	CURS
detect	Integer	inputcode,millisecs (ii)	Detect input during specified time	TDET
except	-	-	Deal with exception (error)	XCPT
halt	-	-	Halt the program	HALT
input	String	prompt (s)	Give prompt and read input line	RDLN
keybuffer	-	size (i)	Create keyboard buffer of given size	BUFR
keyecho	-	true/false (b)	Turn keyboard echo to console on/off	KECH
pause	-	milliseconds (i)	Pause program for specified time	WAIT
read	String	maxchars (i)	Read string from keyboard buffer	READ
reset	-	inputcode (i)	Reset input status to default -1	ICLR
status	Integer	inputcode (i)	key or mouse status, shift/ctrl/alt etc.	STAT
time	Integer	-	Return time since start (millisecs)	TIME
timeset	-	milliseconds (i)	Set time counter to value	TSET
try	-	-	Try code while catching errors	TRY
			File processing	
chdir	-	directory name (s)	Change current directory	CHDR
checkdir	Integer	filename,code (si)	Create/delete/check directory	DIRY
checkfile	Integer	filename,code (si)	Create/delete/check file	FILE
eof	Boolean	file handle (i)	Test for end of file	EOF
eoln	Boolean	file handle (i)	Test for end of line in file	EOLN
fclose	-	file handle (i)	Close file	CLOS
fcopy	Boolean	oldname,newname (ss)	Copy file	FMOV
finddir	String	dirname pattern,handle	Find first directory matching pattern	FDIR
findfirst	String	filename pattern,handle	Find first file matching pattern	FFND
findnext	String	file handle (i)	Find next file/directory matching patterr	FNXT

Here the "Type" column is relevant for *functions* that return a value, "Parameters" specifies the names and types of parameters ("i" integer, "s" string, "b" Boolean), and "PCode" indicates the corresponding machine code instruction. The last of these is relevant only to those who wish to explore "Machine Mode" with its ability to see inside the virtual machine on which the *Turtle System* programs run. These facilities are not covered in the current series of Turtle Python teaching materials, but will be the subject of other materials – aimed at a A-Level audience – that are under development.

6. The Example Programs, and Configuring the System

We have already seen one example program, but there are over 100 others, ranging from simple and more advanced graphics, to algorithmic structures and commands, interactive and animated programs, AI and strategy games, file handling, advanced algorithms, and a wide range of computer models from maths, physics, biology and other fields. Many of the *Turtle System* teaching materials are structured around these, but as you progress, you are likely to find that more and more of these programs become accessible to you just by loading, examining and running them.

If you are a class teacher, then you might feel that having all these example programs available to your students is too distracting. But fortunately, this is one of very many things that can be configured as you wish, by going into Expert mode and either selecting "Lock or restrict system setup" from the "Options" menu (or, more complex and versatile, using the two last options from the "Examples" menu). Quite generally, you can set up the system as you wish in lots of respects, so that for example it starts in a particular programming language, with certain programming options selected (e.g. default Python indenting), and with particular menus, tabs, or example programs either available or hidden etc. If you now go to the "Options" menu, fix the desired default Mode (with "Lock or restrict system setup") and finally save a *Default.tgo* file to the Turtle directory (with "Save current settings to options file"), then your desired setup will take effect whenever the system starts up. Among the possibilities provided by the "Lock or restrict system setup" dialog is to hide the facilities which enable these configuration options to be changed. If the *Turtle System* is being run from a school network directory whose files cannot be edited or deleted by the students, then they will be unable to override the configuration set by *Default.tgo* when the system starts up. Details of all this are available in the *User Guide for Teachers* available on the *Turtle* website through the "Documentation" link.