Turtle basics

The numbers on the left are line numbers and don't need to be typed in. The codes on the right are Python commands and they have to be copied exactly as written. Make sure you copy all the symbols, and that everything is in the correct case.

```
from turtle import *
forward(100)
done()
```

Useful turtle commands

The parts of the line that come after a # are called comments, and are ignored by Python. You don't need to type them in.

```
left(90)        #turn left by 90 degrees
right(120)      #turn right by 120 degrees
pencolor("red") #change pen colour to red - note 'colour' is spelt 'color'
pensize(5)      #change the pen thickness to 5
penup()         #lift the pen up so we can move without drawing
pendown()       #put the pen down again
speed(0)        #draw as fast as possible
```

Important data types

- An int (short for integer) stores an integer number, eg. 5
- A float (short for floating point) stores a non-integer eg. 5.111
- A string stores some text eg. "hello"

Operators

Basic mathematical operators:

```
a + b
a - b
a * b
a / c
```

Conversions

Converting from one data type to another:

```
#converts a string to a float
converted = float("1.54")

#converts a string to an int
converted = int("3")

#converts a float or an int to a string
converted = str(2.5)
```

Variables

We can store some data in a variable using the = operator:

```
a = 10
b = "my stuff"
```

Then we can recall them by using their names:

```
print(a+5)
print(b)
```

And also update them:

```
a = a + 5
b = "hands off " + b
```

User Input

We can ask the user for a string (and store it in a variable called `answer`):

```
answer = input("what is your name?")
```

Conditionals

This is how we write a program that can respond differently depending on the conditions in which it runs:

```
time = 12
if time < 12:
    print("morning!")
elif time >= 12 and time < 18:
    print("afternoon!")
else:
    print("evening!")
```

We have these to choose from:

- == exactly the same as. 2 equals signs for comparison, 1 for variable assignment.
- >= more than or equal to
- > more than
- <= less than or equal to
- < less than
- != not equal to

Loops

To loop forever:

```
while True:
    print("hello!!")
```

To loop a certain number of times we can use `while`. `while` will only loop the code after the `:` while its condition is True:

```
#make a variable to keep count
loops = 0

#keep running the code while the loops variable is less than 10
while loops < 10:
    print(loops)

    #increase the loops variable by 1
    loops = loops + 1
```

Libraries

We can use libraries to get extra functionality in our programs. For example, to sleep for some time:

```
import time
time.sleep(5)
```

Or to get a random number between 1 and 10:

```
import random
random.randint(1,10)
```

Functions

If we are copying and pasting the same code over and over, we can use a function to save time and improve readability. In this example the function is called `my_func` and it needs 2 arguments (arg1, and arg2).

- When we call the function we need to provide the right number of arguments
- We can only call a function after it's been defined with the `def` keyword

```
#define the function
def my_func(arg1,arg2):
    print(arg1 * arg2)

#call it
my_func(10,100)
```

Lists

Creating a List

```
my_list = ["rabbit", "cat", "dog", "hamster", "budgie"]
```

Print a list item

```
print(my_list[0])
```

Important: list indexing starts at 0, not 1.

Print out the list one item at a time using a `for` loop

```
for item in my_list:
    print(item)
```

Change a list item

```
my_list[0] = "goldfish"
```

Add an item to the list

```
my_list.append("rabbit")
```

Delete an item from the list

```
del my_list[5]
```

Basic file I/O

Opening files for read or write

`open()` takes 2 parameters: the name of the file and the read or write mode. Some important modes are:

- 'w' : open the file for writing (will delete any contents first)
- 'r' : open the file for reading
- 'a' : open the file for writing, but will append new data to the old

`open()` returns a 'file handle' which is why I shorten it to `fh` in the examples. The file handle is the bit that lets us read or write.

Writing files

`write()` takes one parameter: the data to write. It can be anything. If you want to write strings on separate lines you'll need to add the newline character `\n` or all the lines will be joined together.

`close()` closes the file handle, which is good practice but often not necessary; as Python will close all open files when the program ends.

```
# open the file for write
fh = open("file.txt", 'w')

# in a loop:
loops = 0
while loops < 10:
    # write hello 10 times. The `\n` is to make each line separate
    fh.write("hello\n")
    loops = loops + 1

# close the file
fh.close()
```

Reading files

`readlines()` reads the whole file into a list, with each line as a separate element in a list.

```
# open the file for read
fh = open("file.txt", 'r')

# read all the lines into a list
lines = fh.readlines()

# using a for loop, print the lines out
for line in lines:
    print(line)
```

When the file is printed, there is an extra line in between each ‘hello’. This is because the `print()` function adds its own newline character to the string, which already has its own newline - meaning 2 newlines get printed.