An Introduction to Programming using Python

Mark Hepple

Department of Computer Science
University of Sheffield

January 2014

Aims of this session

Aims

- What is programming?
- To introduce the Python programming language
- To give you some hands-on programming experience
- Simple ideas - loops, functions, variables, decisions –
  - ... giving rise to complex *seemingly* intelligent behaviour

Structure

- 10 minute intro
- hands-on programming session
- short debriefing session

Programs and Algorithms

- A computer *program* is a set of instructions that tell a computer how to carry out a task.
- The instructions are written in a special *formal* language.
  - a programming language
- In order to solve a programming problem, we need a step-by-step specification of the actions that must be taken to compute result.
  - this specification is called an *algorithm*
- A algorithm should be:
  - *precise* and *unambiguous*,
  - *correct*, i.e. finish and deliver correct result,
  - *efficient*, but this depends on task.

Algorighms – Examples

- Algorithms can be expressed in English-like '*pseudocode*.'
- Task: making a cup of (instant) coffee

```
1. Fill kettle
2. Boil kettle
3. Put spoon of coffee in cup
4. Fill cup (nearly) with water from kettle
5. Add a dash of milk
```

- This algorithm is just a single fixed sequence of actions.
- Can handle more complex tasks by allowing:
  - *conditionals*: actions that happen only under certain conditions,
  - *loops*: (groups of) actions that repeat over until result achieved.

Algorighms – Examples

- Task: supermarket shopping on a budget!!

```
1 Get a trolley
2 WHILE there are items on shopping list
  2.1 Read first item on shopping list
  2.2 Get that item from shelf
  2.3 IF item costs less than £3
    2.3.1 Put item in trolley
  2.4 ELSE
    2.4.1 Put item back on shelf
  2.5 Cross item off shopping list
3 Pay at checkout
```

High-level programming languages

- *High-level programming languages* use English-like syntax.
  - easy for us to understand, but cannot be understood directly by a computer
- Computers understand a low-level binary language called *machine code*.
- A program written in a high-level language must be converted to machine code before it can be executed.
- Some languages are *compiled*, e.g. Java.
  - high-level program converted to machine code in one go, by a *compiler*
- Some languages are *interpreted*, e.g. Python.
  - each line is converted to machine code then executed in turn

Working with the Python Interpreter

- Python is an *interpreted language*
  - can be accessed directly in *interactive mode* (shell mode)
- Can access python in shell mode by running IDLE
  - IDLE is Python's *Interactive Development Environment*
- Interactive shell at start up might look like:

```
Python 2.7.2 |EPD 7.1-1 (64-bit)| (default, Jul ...
[GCC 4.0.1 (Apple Inc. build 5493)] on darwin
Type "copyright", "credits" or "license()" for ...
>>>
```

- In the above, ">>>" is the Python prompt
  - can enter instructions 1 by 1 at prompt
  - Python will convert and execute them

Working in the Python shell

- Simplest use: can enter arithmetic expressions
  - Python will evaluate them, and print result
  - bit like a glorified calculator

```
>>> 2 + 3
5
>>> 2.9 + 3.2
6.1
>>> 3.2 / 2.0 + 7
8.6
>>> (3.2 / 2.0) + 7
8.6
>>> 3.2 / (2.0 + 7)
0.35555555555555557
>>>
```

Storing code for reuse

- There are many tasks that we may want to repeat
  - commands entered to Python shell are 'lost to time . . . '
  - better to *store* the commands in a file for later reuse
- Example: we can use IDLE as an editor, to create a file with the following contents:

```
# Converts distance in miles to kilometers
miles = 22.7
kilometers = (miles * 8.0) / 5.0
print "Converting distance in miles to kilometers:"
print "Distance in miles: ", miles
print "Distance in kilometers:", kilometers
```

  - such a file is called a *code file*, *script* or *program*
  - the line beginning "#" is a *comment* – it is ignored
  - in IDLE, we can run the code just by pressing `F5` (provided edit window is *frontmost* window)

Storing code for reuse

- Example continued . . .
  - For this example, when the code is run we will see the following output appear in the interpreter window:

```
>>> ==================== RESTART ====================
>>>
Converting distance in miles to kilometers:
Distance in miles: 22.7
Distance in kilometers: 36.32
>>>
```

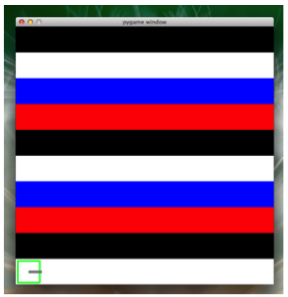
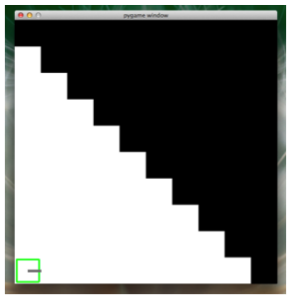
  - we could now use editor to set different values for the miles variable, and re-run to compute a range of results

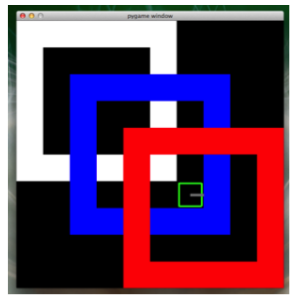
Introducing Robby

- We have provided you with a bit of software that will simulate Robby the robot painter.
- Robby *moves on a grid* and can perform some *simple actions*.
  - turn left or right
  - move forward
  - paint the square
  - look at the colour of a square
- We will use Robby to introduce *basic programming concepts*.
  - loops – repeating things
  - variables – remembering things
  - functions – defining complex instructions in terms of simpler instructions
  - making decisions – e.g. reacting to our environment

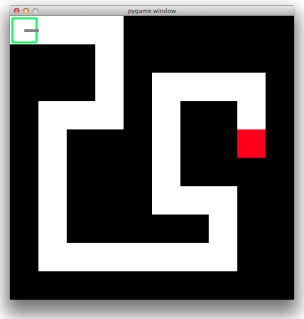
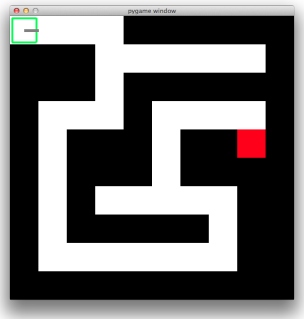
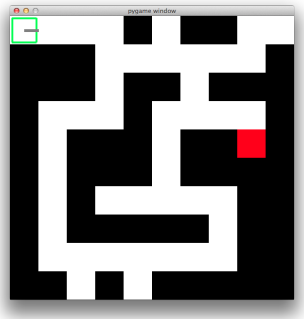
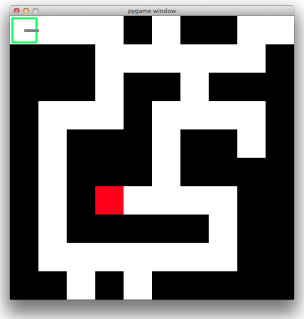
What you're asked to do...

- The session has two main components:
  - Painting patterns of increasing complexity.



  - Using Robby's eyes to navigate a maze.

- Follow the lab sheet but ask for help when you get stuck!