**Notes on programming to support this teaching and learning programme**

**Ruth Davey**

When teaching programming, it pays to support your students to use conventions and develop good habits. They will then always do things this way and save a lot of time later.

Create your own classroom conventions based on the Python (or the language you are using) conventions and common sense.

Python is a powerful language and a good teaching language, but it does allow you to do things which are not considered good practice.

Support your students to:

* start each program with the comment block shown below
* plan their program (even a very small one)
* plan how to test it (BEFORE they code it)
* comment their code meaningfully
* use accepted conventions
* use your conventions
* use some form of version control
* use well chosen, descriptive variable names (self-documenting)
* structure their program correctly
* break larger tasks down into smaller tasks, code these one at a time and test thoroughly BEFORE coding the next small task. Save as a new version BEFORE coding the next section. This will help you to never lose ALL your work. You will always have a working version to go back to.
* NEVER name a program with the same name as a python module or function. It is possible to do this, but definitely not wise to do so. It leads to unexpected error messages.

**My version of correct layout for a program**

* Comment block identifying the program, author and date created. I do it like this:

#Program: (program name)

#Author: (your name – you wrote it)

#Date: (date you started – today)

#Version: (version # - start at 1)

* any imports or include statements
* constant variables
* (any classes – usually senior levels)
* functions used by the program
* main program

I have found that one good way to get students to plan, and show this plan, for their code is to get them to create pseudo code as comments in their first version. They can then add in the code that performs that portion of the program. The program is thus also commented sensibly from the beginning.

It also pays to teach good file and folder management throughout the year. Model this with the examples by suggesting where students should save their work and what they might name their folders and files.

Each new project or exercise should have its own folder. It makes student work neat and easy to find. It also entrenches good habits.

I find that making students type in the code themselves rather than copy and paste helps them to learn to code. Once they have mastered the basics, code starters can be supplied so that all they need to do is type in or create on their own the new section. For this reason I usually supply students with images showing the required code. Code on paper will also work.

**Some Python conventions:**

These make the code easy to read and understand.

1. use lower case for variable names
2. use underscore for multi-word names eg total\_cost
3. use UPPER CASE for constant variable names eg TAX\_RATE
4. Class names should start with a Capital letter eg Pet\_rock.
5. use single word names for programs (underscores used to join multiple words)

**Naming Variables**

* All letters and numbers can be used
* Must start with a letter
* Symbols can't be used (exceptions below)
* Spaces can't be used
* Use an underscore (\_) instead of spaces
* Python is case sensitive - upper & lower-case letters make a difference
* Don't use Python reserved words
* Don't use Python function names, method names or module names as variable names

**Conventions**

* Use lower case for variable and function names
* Make names meaningful
* When more than one word required, separate them with \_ (underscore) eg first\_name (preferable to firstName)
* Use UPPER CASE for Constant variable names eg TAX\_RATE = 0.15

**Error that are easily made**

* Misspelling a variable name - ALWAYS spell the name the same way

**Smart output - in the print statement**

, separates different items

\n sends next output to a new line

\t tabs before next output

sep="-" separates items with - between them

sep='\n' separates with a new line

end=''." adds a full stop at the end of output

**Data Types**

Integers - numbers without decimal point

Floats - numbers with decimal points

Strings - include letters, numbers, spaces, symbols inside quote marks

Booleans - True or False (start with Capital letter)

**Get Python to tell you the data type of a variable**

type(…..) - will report the class type

eg type(name)

Can convert types

int(apple) - converts the contents of variable "apple" to integer (if possible)

str(apple) - converts the contents of variable "apple" to string

**Notes on Strings**

* Strings are surrounded by quote marks - either single ' or double "
* If you want to use either of these in your string, surround it with the other or use an escape

Example

message = "I'm enjoying learning Python"

or

print('I\'m enjoying learning Python')

**Some useful string methods**

* name.title() - makes the string stored in the variable name have the first letter of each word as a capital letter
* name.upper() - everything in the string stored in name upper case
* name.lower() - everything in the string stored in name lower case
* name.rstrip() - strips all blanks to the right of the string from the string
* name.lstrip() - strips all blanks to the left of the string from the string
* name.strip() - strips all leading and trailing blanks from the string

**NOTE:** none of these methods change the contents of the variable itself, only how it looks

**Concatenation**

Joining two strings together to make a new string

use " + " between the two strings

**Lists**

* a collection of items in a particular order (can be changed)
* anything can be placed in a list
* hint: always make the name of your list a plural word as lists usually contain more than one variable
* [ ] indicate a list
* individual items in the list are separated by ,

example

cars = ['Toyota', 'Mazda', 'Mercedes']

* to access individual items in a list, use its index (always counted from 0)

eg print(cars[0]) should print out Toyota

* to access the last element in a list, or elements from the end, use negative indexes

eg print(cars[-1])

* to change an item in a list, use an assign statement with the index number for that item

eg cars[0] = 'Holden'

* to add an item to a list, use the append() method

eg cars.append('Ford') will add Ford to the list cars

* to insert an item in a list use the insert() method

eg cars.insert{1, 'Zepher') adds this make of car after the first element of the list

* to remove an item from the list use the del statement

eg del cars[3]

**Tuples**

* Like lists, and ordered data structure that is immutable (can't be change once created)
* Once objects assigned to the tuple, they cannot be changed
* Think of these as constant lists
* Indexed data type (starting at 0)

**Dictionaries**

* Unordered data structure
* Set of key/value pairs
* Each key has a value associated with it (value can be an object)
* Dictionaries are mutable (can add, remove and change contents)

**Sets**

* Data structure without duplicates
* Unordered collection of related objects

**Functions**

students at this level need to be able to use functions, and should know how to send a value into a function and how to return a value.

* Functions are used to:
	+ allow for code to be reused in a program
	+ make a program easier to read
* values are passed into functions using arguments and received by the function in the function parameters
* a function can return a value but does not need to do so
* use same naming conventions as for variables
* always define the function BEFORE calling it

**Classes**

Students at this level are not required to know how to create their own classes. They are not even required to use them at all, but to create exciting games with sprites that move etc using Pygame and Arcade, they will need to know how to work with classes. I suggest that you share the code with them for such example classes, have discussions in class about how they work and then allow students to modify and use this code in their games. They will not be graded on this code as it is not their own. They will have sufficient scope for writing their own code when modifying and using such example code. Evidence from their early class exercises will also help them to achieve the programming standard. Class exercises are part of the process, so are vitally important.