Notes on Object-Oriented Python Programming

1. Introduction

A limited and haphazard survey of the extensive literature on teaching object-orientation has suggested three papers to read. Sanders *et al.* [1] give two lists, one of OO concepts and the other of misconceptions. These lists, reworded, form the most important part of these notes in Section 3. Miller *et al.* [2] identify some problem specific to Python: these are addressed in Section 2 and also used to adapt the checklists. Finally, Sanders *et al.* [3] (the same team as [1]) use concepts maps to assess the depth of understanding of OO concepts: the technique is interesting and provides further support for the existence of misconceptions.

Most of the literature relates to teaching OO at University and assumes that the objective is to achieve proficiency writing OO programs. This may not be the objective for all A-level syllabuses.

2. Issues for Teaching OOP

2.1 General Issues

Some general issues for teaching OOP are described below.

1. *Relationship to computational thinking*.
    a. OO achieves decomposition in the form of program organisation. OO can therefore be presented as an extension to program organisation using functions / methods (so called 'functional decomposition'). It has the same difficulties: the additional organisation is a barrier at first before becoming an enabling capability.
    b. Abstraction is relevant, as a class should represent a concept from the problem. See next point and also notes on encapsulation issue in Python.

2. OO modelling and OO programming.
    a. A major advantage claimed for OO for programming (OOP) is that both the problem can be analysed using object concepts (OOA) and the program can be structured using the same concepts. However, there are some practical implications of this.
    b. The vocabulary differs between OOP and OOA. For example, attribute, fields and member variables are all roughly the same.
    c. Statements we can make about OO have to be understood in context, either OOA, OOP or both. For example:
        i. An object is a real-world entity in the problem domain (OOA).
        ii. An attribute belongs to a class (mostly OOP).

       Since the context is not always apparent, this can cause confusion. Some statements can be ambiguous: for example 'A book is a relevant object in this domain' suggests that 'Book' is a suitable class.
    d. To what extent and how do we teach OO modelling with programming? The modelling could be taught first or afterwards. Including some modelling (or analysis) may help to develop problem solving and abstraction. However, it is very abstract if taught independently of programming. An intermediate

approach is to include some OOA alongside programming (e.g. ideas of class responsibility and identity) but without developing OOA fully.

2.2 Python Specific Issues

Python's lack of static checks means that there are fewer errors, and most errors are runtime. See [2, section 3.2.2] for an example of what can happen. Some specific difficulties are described below.

1. *Definition versus declaration*.
    a. In Python, variables are defined rather than declared. A common pattern is to assign default values to all attributes in the constructor, so the constructor in effect 'declares' the attributes. The constructor should be introduced early therefore.
    b. Giving types to attributes (in a static language, not Python) underlines the possibility of using an object (of another class) as a value of an attribute to create a 'has-a' relationship between classes (also called a 'peer' class). The same relationships exist in Python but without declaration they can easily be overlooked.

2. *Use of self*.
    a. A method called with one parameter is declared using two: conventionally the first is called 'self'. This creates a gap between the syntax of the call and declaration.
    b. In a method of a class C, an attribute of C must be referenced using self, again creating a gap. Unfortunately, omitting this may just create a local variable in the function and the error may not be obvious.

3. *Lack of encapsulation*.
    a. Encapsulation is the flip side of abstraction and often presented as an advantage of OO programming. However, Python does not enforce any encapsulation and the intended use of variables (or methods) can only be shown using naming conventions. Understanding that a variable is intended to be 'private' can help develop an understanding of abstraction.
    b. Python classes are entirely open; when you understand how it works you find not just that the attributes can always be accessed by the 'user' of an object, but new attributes can be added too. This can be ignored but the practical implication is that there are more ways to 'get it wrong' without getting an error.

4. *Functions as members*.
    a. In Python, functions are members of the class and therefore can be accessed like attributes. This is usually done by accident when the brackets are omitted from a function call, and this does not give an error. See [2, Section 3.2.3] for an example.

3. Concepts and Misconceptions

3.1 Concepts

The following table of concepts is adapted from [1].

Concept		Prerequisite Knowledge
Basic mechanics	Calling a method of an object.	Calling a function
	Class as a template for data: "what the object knows".	Variables
	Class as a collection of methods: "what the object does"	Functions and parameters
Constructors	Definition and use of constructors	Functions and parameters
Interaction	Objects as values, including in a list (etc)	Class declaration and construction
	Object as attribute value (has-a relationship)	
	Object passed as parameter to constructor	
	Object passed as parameter to method	
Abstraction and modelling	A classes represents an abstract concept in the problem domain	Class declaration and construction
	Methods have parameters	
	Constructor has parameters	
Inheritance	Subclasses defined. Attributes and methods added.	Class declaration and construction
	Superclass constructor called	
	Common code moved up hierarchy	
	Methods overridden	

3.2 Misconceptions

The following table of misconceptions is adapted from [1].

Misconception	Possible Evidence
Attributes defined or referenced in the wrong scope Note: use of self is mostly done by following rules rather than a deep understanding.	Incorrect use of self, usually omission.
Confusion between class and object (or instance)	The program is run without any objects being created. Confusion that nothing happens. Classes always have only a single instance (object) Inheritance used instead of object creation: e.g. Bob as a subclass of person rather than an instance.
Confusion between class and attribute	Many classes, all very simple.

Misconception	Possible Evidence
Objects only contain data	No classes with methods other than constructors or get and set.
Objects do not interact	Code in single class instead of several classes Classes defined but not imported Work in methods always done by assignment Objects never passed as parameters Objects passed as parameters but not used
Believing objects are copied not referenced	Aliasing errors Constructing a new instance rather than updating an existing one
Not understanding that a class creates a scope	Methods/variables in different classes always have different names

4. Bibliography

1. Sanders K, Thomas L, Sanders K, Thomas L. *Checklists for grading object-oriented CS1 programs*. In proceedings of the 12th annual SIGCSE conference on Innovation and technology in computer science education - ITiCSE '07. City: New York, New York, USA. Publisher: ACM Press. 2007 vol: 39 (3) pp: 166, DOI 10.1145/1268784.1268834
2. Miller C, Settle A, Lalor J. *Learning Object-Oriented Programming in Python*. In proceedings of the 16th Annual Conference on Information Technology Education - SIGITE '15, City: New York, New York, USA. Publisher: ACM Press, 2015 pp: 59-64, DOI 10.1145/2808006.2808017
3. Sanders K, Boustedt J, Eckerdal A, McCartney R, Moström J, Thomas L, Zander C. *Student understanding of object-oriented programming as expressed in concept maps*. ACM SIGCSE Bulletin, 2008 vol: 40 (1) pp: 332, DOI 10.1145/1352322.1352251