# NOT "An Object To Think With" For Novice Programmers

**Dermot Shinners-Kennedy**
Computer Science and Information Systems Department
University of Limerick
Limerick
Ireland
dermot.shinners-kennedy@ul.ie



## Abstract
We are now in the computer rich environment that Seymour Papert described more than a quarter of a century ago. However, the effect computers have on how people think and learn is not quite as dramatic as he predicted. In some cases our persistent and pervasive interaction with computing devices has diminished our inquisitiveness, reflection and engagement with thinking, understanding and reasoning about the various concepts and tools we use on a daily basis. Focusing on three foundational concepts of computing in general and programming in particular we consider the difficulties evidenced in novice programmers' inability to utilise concepts they depend on in their everyday activities. Despite exhibiting a high degree of competence and expertise with these concepts in everyday tasks their ability appears to desert them when placed in a programming context.

## Author Keywords
Foundational concepts; learning difficulties; novice programmers; teaching programming.

## ACM Classification Keywords
K.3.2 [**Computing Milieu**]: Computer and Information Science Education – *Computer science education*.

## Introduction
Over a quarter of a century ago Seymour Papert published his Mindstorms vision of "[a] computer rich

future, a future where a computer will be a significant part of every child's life" and he speculated how that environment might "affect the way people think and learn." Papert believed access to computers would provide opportunities for children to sharpen their "thinking about thinking" and to reflect on their own actions and their own thinking.

We are in that future now. As part of their everyday experiences children, adolescents and teenagers interact with mobile phones, games consoles, music players, TV box sets, text messaging, internet browsers, Facebook, Twitter, and a host of other tools and devices. These devices have more computing power than the first Apollo spacecraft that landed on the moon, and virtually every youngster has access to them. That computing power represents what Papert identified as an object-to-think-with, "[an object] in which there is an intersection of cultural presence, embedded knowledge, and the possibility for personal identification" (Papert 1980).

As part of their daily interactions with technology children have the opportunity to acquire and apply many of the concepts that are fundamental to programming knowledge acquisition. In many cases they successfully acquire the concepts but too often they fail to apply that knowledge in new situations they encounter. The challenge for tutors is to unhide this knowledge that is hidden in plain sight and make it a productive tool for the learner.

In the following sections a series of everyday concepts is sketched out and the connections to programming concepts highlighted. The simplicity of many of the concepts only adds to the irony of their status as barriers to programming knowledge acquisition. The phrase "hidden in plain sight" is used to emphasise the fact that these concepts are used every day by primary and secondary school students but appear elusive at precisely the time the student is seeking to find a solution to a problem that could benefit from the application of the concept.

## Constructors

Posing the following conundrum to a group of young mobile phone users is sometimes viewed as an insult to their intelligence. The conundrum asks them to explain what has gone or is going wrong with a phone that doesn't have the correct date and time, is not able to connect to the internet and has no entries in the contacts list. The group will very quickly point out that the phone has not been set-up properly. The data and time have not been set to the owner's locale, the internet settings have not been modified to connect to the owner's provider and the owner has not added the names and numbers of their preferred contacts. When asked if this is necessary for every phone that is purchased the group will look at the person posing the question with disbelief and answer yes, with varying degrees of derision evident in the tone of the yes.

Subsequent discussion with the group establishes that this type of activity (i.e. the requirement to configure or set-up or prepare for use) is commonplace and is a necessary part of the utilisation of all sorts of gadgets and devices. The group's unequivocal commitment to this position can be verified by asking if it is necessary to set-up an MP3 player; a smart phone; a new computer. The implications of not doing the set-up are clear - the device will not work properly.

When this idea is developed more the group will often accept that the device will actually work properly but because it hasn't been configured properly it will not work in the way required by the user. So the device may, for example, keep the time correctly - ticking by

each second and moving correctly on the minute and the hour - but it does not have the correct time required by the user and as a consequence is worthless to the user. This leads to the group's acceptance that despite working as claimed by the manufacturer, despite correctly doing the operations it claimed to provide, the device is worthless because it did not start from the correct initial settings or value.

Even novice programmers who accept this line of reasoning have difficulty with the concept of initialisation and its pivotal role in the development of a correct solution to a problem. The computer science education literature contains many reports of these difficulties. For example, in object-oriented systems the constructor method is responsible for effecting the set-up of an object. Despite the proximity of its responsibilities to the requirements of device set-up it is a persistent source of difficulties for novice programmers (Ragonis and Ben-Ari 2005). Novices programmers' existing knowledge of the importance of initialisation evades them when they need it most.

## Lists

Mark Twain once observed that "It ain't what you don't know that gets you into trouble. It's what you know for sure that just ain't so!" By way of example, consider the concept of a list, a universally applicable everyday concept. Making, modifying, using or discarding a list would be considered a trivial task by even very young children.

The car manufacturer Mercedes Benz has a TV advertisement for its cars which features lists[1]. The ad includes the following observation "There are 1000's of different lists and all have something in common. On a list there is always a first." This is a common misconception. You can very easily confound someone by asking them to give you an example of an empty list. The concept of an empty list is counter-intuitive for most people. Who would have a list with nothing in it? How could it possibly be considered a list if it has no entries? However, if prompted, most people will accept the concept of an empty To-do list. Similarly, if you ask a group of people to provide you with a list of the people they have murdered you would hope that the list would be empty!

When asked, most students will claim not have encountered self-organizing lists, last-in first-out lists, or a list that could be empty. However, a little prompting allows them to realize that the text prediction feature in their phone initially displays word suggestions based on general usage but over time preferences the words that have appeared frequently in text messages. How the phone handles incoming messages by placing the most recent as the first reveals the stack-based nature of the incoming message list.

Lists are the source of very many conceptual difficulties encountered by students of computing. Ironically, the students' everyday list handling capabilities are quite sophisticated. Many of the difficulties arise from the possibility of the list being empty, an aspect of the everyday conception of list that is so layered or compressed that it is rarely if ever considered a possibility. Unfortunately, languages like Scratch 'hide' the nuances of lists by suppressing list handling errors.

## Where are you? I'm in the cloud

Mobile phones users often begin phone calls with the question "Where are you?" Because the phone is mobile the location of the person answering it is variable. The

[1] Available at https://www.youtube.com/watch?v=jstevXz1880 (last accessed 1 May2015)

beauty and power of this arrangement is that the caller does not have to worry about the receiver's location. Simply having the receiver's phone number provides a line (literally) of access to them. It is the task of the phone network to determine where the receiver's phone is physically located on the planet and to action the appropriate connections to make the call possible. The same resources are also required to ensure the call can continue until the parties wish to terminate it.

URL's offer similar access to computer-based resources instead of mobile phone users. Again the physical location of the machine identified by the URL is of no interest to the user. Simply typing the URL into a browser causes the appropriate resources to be activated to locate and connect the user to the site and to continue to maintain that connection until the user decides otherwise. The only perceived limitation on the user is that once they have connected to a site they are only able to use the facilities or operations provided by that site, and no others.

Everyone seems to be comfortable with the idea of cloud computing. The concept is pervasive in popular culture and even very young children can grasp it. Even if they are not aware of the technological implications many seem to have grasped the concept of 'things' being located somewhere in the absence of any knowledge as to precisely where that location is. As long as the mechanism for providing connection and access works as anticipated the details of the actual location become irrelevant.

Programmers use the term pointer or reference to refer to the same type of facility provided by mobile phone numbers and URLs. A pointer or reference identifies a location in the memory of the computer currently storing an object of some type. Using the pointer or reference provides access to the object and the operations provided by the object. In a program that has many types of objects you can have many pointers but when you use a pointer to a particular object you can only use the operations provided by that object. The mechanism is exactly like URLs and web pages.

Thus, despite their apparent comfort with the use of mobile phone numbers, URLs and cloud computing identifiers as a way of accessing resources in an unspecified location many novice programmers experience pointers as one of the most difficult concepts in computing.

## Conclusion

Tutors often forget that what appears to them to be a simple concept may be problematic for learners. This leads them to assumptions about the learners that may not be realised. In contrast, learners often fail to apply knowledge that they already have to new situations that they encounter because the knowledge is so rehearsed and embedded in the original context that they are unable to appreciate the connection. This leads the learner to erroneously believe that they have failed to acquire new knowledge and that the new context is beyond their comprehension.

## References

Papert, S. (1980) Mindstorms: children, computers, and powerful ideas, Basic Books, New York, USA

Ragonis, N. and M. Ben-Ari (2005). "A Long-Term Investigation of the Comprehension of OOP Concepts by Novices." Computer Science Education 15(3): 203-221.