



### LEAVING CERTIFICATE COMPUTER SCIENCE

National Workshop 1



This work is made available under the terms of the creative Commons Attribution Share Alike 3.0 License <u>http://creativecommons.org/licences/by-sa/3.0/ie/</u>. You may use and re-use this material (not including images and logos) free of charge in any format or medium, under the terms of the Creative commons Attribution Share Alike Licence. Please cite as: PDST Leaving Certificate Computer Science National Workshop 1 Participant Booklet, 2023.



### **Table of Contents**

Session 1: Introduction to LCCS	4
Meet the Team	4
Key Messages	5
Think - Pair - Share	6
Ice Breaker	
Growth Mindset	11
Culture and Expectations	13
LCCS CPD Timeline	14
LCCS Curriculum Specification	17
Session 2: Learning Outcomes and the ALTs	
Learning Outcomes	
Group Activity 1: Linking the Learning Outcomes	
Constructivist Pedagogy	
Group Activity 2: Investigating the ALTs	
Group Activity 3: Expanding Your Idea	
Session 3: Computational Thinking Activities	
Activity 1: Aspects of Computational Thinking	
Activity 2: A Familiar Task	
Activity 3: Abstraction	
Activity 4: Adapting the Monty Hall Problem	
Activity 5: Computational Thinking Methods	
Session 4: Teaching & Learning Programming	
Learning Challenges faced by Novice Programmers	
Programming Pedagogies: Warmup Activity	
Programming Pedagogies: Peer Instruction	
Programming Pedagogies: Parson's Problem	
Programming Pedagogies: PRIMM	



### **Session 1: Introduction to LCCS**

### **Meet the Team**



Joe English joeenglish@pdst.ie



Sinéad Crotty sineadcrotty@pdst.ie



Irene Stone irenestone@pdst.ie



Tony McGennis tonymcgennis@pdst.ie



Neil Butler neilbutler@pdst.ie



Helen Van Eesbeck helenvaneesbeck@pdst.ie Post-Primary STEM Team Leader





### Key Messages

	Notes:
There are many ways to use the LCCS specification	
	Notes:
The learning outcomes (LOs) are non-linear	
ALTS	Notes:
The Applied Learning Tasks (ALTs) provide an opportunity to teach theoretical aspects of LCCS.	
ili	Notes:
LCCS can be mediated through a constructivist pedagogical approach	
fo V 6- in A 6 P	Notes:
Digital technologies can be used to enhance collaboration, learning and reflection.	
	Notes:
LCCS is a subject for ALL	



### Think - Pair - Share

Question	What I thought	What my partner thought	What we will share
What is Computer Science?			
Who is Computer Science for?			



### **Think-Pair-Share Square Strategy**

This strategy allows you to quickly engage the whole class without losing any time moving furniture or formulating groups. Think-Pair-Square-Share is a series of steps that enables the students to move through the stages of individual work, paired work and group work before feeding back to the whole class very simply.

Think: The students spend time in silence writing or thinking about their own ideas.

Pair: Students turn to the person next to them to discuss their ideas with a partner.

Share: Students share their answers with another group

**Square:** Two pairs work together as a new group to complete the task of agreeing on a response from the first two answers that the pairs have come up with. They also elect who will be speaking. This stage is crucial for extracting the high level explanation behind why an answer was chosen. This reduces the amount of answers that a teacher has to elicit from a class. It helps promote student learning as students discuss and teach each other.

## Notes





- Computer Science is suitable for all.
- Computer Science can be a fun and engaging way to learn about the world we live in
- Computer Science is both innovative and collaborative people work together to come up with solutions that can make a real difference to society
- Computer Science involves design thinking and creativity
- Computer Science is especially suitable for people who are curious and logical and want to learn more about the world around them and develop ways to shape society
- Computer Science is equally suitable to girls and boys
- Computer Science is equally suitable to people who are particularly strong at maths and those with an average ability in maths
- Computer Science needs diversity, not stereotypes!
- Computer Science opens up opportunities in almost every profession imaginable
- Computer Science is relevant to all careers/in all walks of life



Computer science is no more about computers than astronomy is about telescopes, biology is about microscopes or chemistry is about beakers and test tubes. Science is not about tools. It is about how we use them, and what we find out when we do.

— Edsger Dijkstra —

AZQUOTES



### Leaving Certificate Computer Science National Workshop 1 – Participant Booklet

Despite having made significant inroads into many traditionally male-dominated fields (e.g., biology, chemistry), women continue to be underrepresented in computer science. students' stereotypes about the culture of these fields - including the kind of people, the work involved, and the values of the field - steer girls away from choosing to enter them.



Students have stereotypes about the culture of computer science and engineering and girls face negative stereotypes about their abilities. Both types of stereotypes signal to girls that computer science and engineering are not appropriate fields for them.

Source: Cheryan S, Master A, Meltzoff A. Cultural stereotypes as gatekeepers: increasing girls' interest in computer science and engineering by diversifying stereotypes. Frontiers in Psychology. 2015;6(49):1–8.

### Notes



### Ice Breaker

My Fav	ourite	Your Favourite (Write your favourite in this column)	Names (Who else's favourite?)
0	Film		
<b>Y</b> @1	Food		
	Team		
5	Song		
	Book		



### **Growth Mindset**

Should you tell your kids they are smart or talented?

Professor Carol Dweck answers this question and more, as she talks about her groundbreaking work on developing mindsets. Watch these two videos to find out more.



### https://youtu.be/hiiEeMN7vbQ

Learn more about the power of "yet" in helping students succeed in and out of the classroom.



The Growth Mindset 378,216 views

https://youtu.be/wh0OS4MrN3E

1 2.3K - 41 49 → SHARE

Sal Khan from the Khan Academy talks with Stanford Professor Carol Dweck about her research on Growth Mind-set.

## Notes





GRAPHIC BY NIGEL HOLMES



### **Culture and Expectations**

What should the culture be in the group?		
What average da you have from each other?		
what expectations do you have from each other?		

### The Role of the PDST





### LCCS CPD Timeline



### Notes



### COMPSCI.IE



https://www.compsci.ie/

### An Integrated Approach to Learning Teaching and Assessment





https://pdst.ie/sites/default/files/Integrated%20Approach\_0.pdf







### **LCCS Curriculum Specification**

https://ncca.ie/media/3369/computer-science-final-specification.pdf



### The Evolution of Computers and Society

https://curriculumonline.ie/getmedia/a5e0d88d-e0f1-43bc-ab68-349b5660fbce/NCCA-The-Evolution-of-Computers-in-Society-LC-SC.pdf





### LCCS Curriculum Specification

Introduction
Senior Cycle
Computer Science (Rationale/Aim/Objectives)



Related Learning
Structure of Leaving Certificate Computer Science
Key Skills of Senior Cycle



Teaching and Learning (ALTs) Teaching and Learning (Differentiation) Assessment



### Final Reflection (3-2-1)

Complete the 3-2-1 reflection with regard to the LCCS specification.

List three things you learned
1.
2.
3.
List two things you'd like to learn more about
1.
2.
One question you still have
1.



### **Session 2: Learning Outcomes and the ALTs**

### **Learning Outcomes**

Learning outcomes have become ubiquitous within worldwide curriculum policy in recent years. This move comes with many potential benefits, as it shifts the focus from providers to users of education, and it introduces a common language, addressing issues of progression, transparency and equity (CEDEFOP, 2009).

### (Mark Priestly, University of Stirling)

Benefits of Learning Outcomes for Teachers				
Effective course design	• By keeping learning outcomes front and center, teachers can develop courses in which all aspects of the course, including learning activities and assessments, support what they want students to learn <i>(a)</i> .			
Effective assessment of learning	• Clear expectations make it easier to evaluate students' progress and ensure that assessments are targeting the appropriate level of knowledge or skill <i>(a, b)</i> .			
Better time management	• Well-defined learning outcomes simplify difficult decisions about what content to include and what to omit when preparing lessons and assessments (b, c).			
Improved communication	• Teachers can use learning outcomes to have explicit and constructive dialogues with students about the course and their learning, and with colleagues about the expectations of courses (b).			
Improved teaching experience	• Teachers who use learning objectives report less anxiety, more confidence interacting with students, and use more diverse teaching and assessment approaches ( <i>b</i> , <i>c</i> ).			
<ul> <li>[a] Wang, X., Su, Y., Cheung, S., Wong, E., &amp; impact on students' learning approach</li> <li>[b] Simon, B., &amp; Taylor, J. (2009). What is</li> <li>[c] Reynolds, H. L., &amp; Kearns, K. D. (2017). in the college classroom. <i>College Teac</i></li> </ul>	Kwong, T. (2013). An exploration of Biggs' constructive alignment in course design and its es. Assessment and Evaluation in Higher Education, 38, 477-491. the value of course-specific learning goals? Journal of College Science Teaching, 39, 52-57. A planning tool for incorporating backward design, active learning, and authentic assessment bina. 65, 17-27. Created by Sara M. Fulmer			



Leavin	g Certificate	Computer	Science	National	Workshop	1 -	Participan	t Booklet

	< Lower Order Thinking Higher Order Thinking>				
	1.1. describe a systematic process for solving problems and making decisions	<ol> <li>solve problems by deconstructing them into smaller units using a systematic approach in an iterative fashion</li> </ol>	1.5. evaluate alternative solutions to computational problems	1.7. develop algorithms to implement chosen solutions	
Computational	1.2. explain how the power of computing enables different solutions to difficult problems	1.4. solve problems using skills of logic	<ol> <li>evaluate the costs and benefits of the use of computing technology in automating processes</li> </ol>		
Thinking	1.6. explain the operation of a variety of algorithms	1.9. use modelling and simulation in relevant situations			
	$1.10. \ \mbox{discuss}$ when heuristics should and could be used and explain the limitations of using heuristics				
	1.11. discuss the complex relationship between computing technologies and society including issues of ethics		1.12. compare the positive and negative impacts of computing on culture and society		
	1.13. identify important computing developments that have taken place in the last 100 years and consider emerging trends that could shape future computing technologies		1.16. Compare two different user interfaces and identify different design decisions that shape the user experience		
Computers and	1.14. explain when and what machine learning and AI algorithms might be used in certain contexts				
Suciety	1.15. consider the quality of the user experience when interacting with computers and list principles of universal design, including the role of a user interface and the factors that contribute to its usability				
	1.17. describe the role that adaptive technology can play in the lives of people with special needs				
	1.18. recognise the diverse roles and careers that use computing technologies				
Designing and	1.19. identify features of both staged and iterative design and development processes	1.20. collaborate and assign roles and responsibilities within a team to tackle a computing task	1.23. reflect and communicate on the design and development process	1.22. read, write, test, and modify computer programs	
Development	<ol> <li>1.21. identify alternative perspectives, considering different disciplines, stakeholder and end users</li> </ol>	1.22. read, write, test, and modify computer programs			
Evaluation and	2.20. Identify and fix/debug warnings and errors in computer code and modify as required		2.19. test solutions and decisions to determine their short-term and long-term outcomes		
Testing	2.21. identify limitations in completed code and suggest possible improvements				
	2.22. explain the different stages in software testing				
		2.1. use abstraction to describe systems and to explain the relationship between wholes and parts			
Abstraction		2.2. use a range of methods for identifying patterns and abstract common features			
		2.3. implement modular design to develop hardware or software modules that perform a specific function			
		2.4. illustrate examples of abstract models			
	2.10. explain the common measures of algorithmic efficiency using any algorithms studied	2.5. use pseudo code to outline the functionality of an algorithm		2.6. construct algorithms using appropriate sequences, selections/conditionals, loops and operators to solve a range of problems, to fulfil a specific requirement	
Algorithms		<ol> <li>implement algorithms using a programming language to solve a range of problems</li> </ol>		2.9. Assemble existing algorithms or create new ones that use functions (including recursive), procedures, and module	
		2.8. apply basic search and sorting algorithms and describe the limitations and advantages of each algorithm			
	2.11. describe the different components within a computer and the function of those components				
	2.12. describe the different types of logic gates and explain how they can be arranged into larger units to perform more complex tasks				
Computer Systems	2.13. describe the rationale for using the binary number system in digital computing and how to convert between binary, hexadecimal and decimal				
	2.14. describe the difference between digital and analogue input				
	2.15. explain what is meant by the World Wide Web (WWW) and the Internet, including the client server model, hardware components and communication protocols				
		2.16. use data types that are common to procedural high-level languages.			
Data		2.17. use ASCII and Unicode character sets to encode/decode a message and consider the importance of having such standards			
		2.18. collect, store and sort both continuous and discrete data			
Interactive Information Systems	3.1. understand and list user needs/requirements before defining a solution	3.3. use appropriate programming languages to develop an interactive website that can display information from a database that meets a set of users' needs		3.2. create a basic relational database to store and retrieve a variety of forms of data types	
Analytics		<ol> <li>3.7. use algorithms to analyse and interpret data in a way that informs decision-making</li> </ol>	3.5. structure and transform raw data to prepare it for analysis	3.4. develop algorithms that can find the frequency, mean, median and mode of a data set	
			3.6. represent data to effectively communicate in a graphical form		
Modelling & Simulation	3.10. explain the benefits of using agent-based modelling and how it can be used to demonstrate emergent behaviours		3.9. analyse and interpret the outcome of simulations both before and after modifications have been made.	3.8. develop a model that will allow different scenarios to be tested	
Embedded		3.11. use and control digital inputs and outputs within an embedded system		3.13. develop a program that utilises digital and analogue inputs	
Systems		3.12. measure and store data returned from an analogue input		3.14. design automated applications using embedded systems	

### **Group Activity 1: Linking the Learning Outcomes**

Examine the learning outcomes (LOs) and pick 2 or 3 from different strands that could be experienced together.

Which LOs did you choose?

What learning experience(s) would you use to engage your students with these LOs?

Which other LOs could your students experience during this learning?

How would you know if these LOs have been achieved?



### **Constructivist Pedagogy**

Traditional Classroom	Constructivist Classroom
Curriculum begins with the parts of the whole. Emphasizes basic skills.	Curriculum emphasizes big concepts, beginning with the whole and expanding to include the parts.
Strict adherence to fixed curriculum is highly valued.	Pursuit of student questions and interests is valued.
Materials are primarily textbooks and workbooks.	Materials include primary sources of material and manipulative materials.
Learning is based on repetition.	Learning is interactive, building on what the student already knows.
Teachers disseminate information to students; students are recipients of knowledge.	Teachers have a dialogue with students, helping students construct their own knowledge.
Teacher's role is directive, rooted in authority.	Teacher's role is interactive, rooted in negotiation.
Assessment is through testing, correct answers.	Assessment includes student works, observations, and points of view, as well as tests. Process is as important as product.
Knowledge is seen as inert.	Knowledge is seen as dynamic, ever changing with our experiences.
Students work primarily alone.	Students work primarily in groups.

https://www.thirteen.org/edonline/concept2class/constructivism/index\_sub1.html



### **Applied Learning Tasks**

### **Group Activity 2: Investigating the ALTs**

- 1. Each group is assigned a particular ALT.
- 2. In your assigned groups, discuss and share potential ideas (possible project ideas for students) for your assigned ALT.
- 3. Aim for as many ideas as you can.
- 4. Record your ideas in the shared document under your Group Number can be text/images etc.
- 5. Present ideas to the wider group.

## Ideas for your ALT

### **Group Activity 3: Expanding Your Idea**

- 1. Pick one or two of your ideas from earlier
- 2. Look at your idea again this time you will be given some prompt questions to consider
- 3. Record your thoughts in the shared document
- 4. Present your ideas to the wider group

What teaching & learning strategies could you use?

How would you assess?



Can it be linked to other parts of the course?

### What theory could be taught at the same time?

In terms of planning where in the course do you see this ALT fitting in?



### **Session 3: Computational Thinking Activities**



### **Activity 1: Aspects of Computational Thinking**

Write your own short note on the aspects of Computational Thinking
Abstraction
Decomposition:
Pattern Recognition:
Algorithm Formation:



### Activity 2: A Familiar Task

Consider how you carry out a familiar task. How did you do it? What computational aspects are used? Are there other ways of doing this? Do these have advantages / disadvantages?

Subtraction	
Description / Analysis of Method:	
Describe other methods / give advantages and disadvantages.	



### Activity 3: Abstraction

Name and describe some ways in which Abstraction is used in the design of the London Tube Map.

Examples of abstraction



### Activity 4: Adapting the Monty Hall Problem

Think about a quiz show where the first prize is €1 million and the final contestant has to choose one of three doors, behind one of which is the big prize.

The contestant chooses, say Door A, but, before the door is opened, the quiz show host reveals that there is nothing behind one of the other doors (B or C). The contestant is then invited to change their choice.



What should they do - stick with their original choice? Change their choice? Or does it not matter what they do? Write down your first thoughts and then try experiments in threes to see what pattern emerges.





Use the space below to record results and conclusions of experiments (using Paper cups and a coin).

One person plays the role of contestant, one as host and the third person can record.

# **Results and Conclusions**



### **Activity 5: Computational Thinking Methods**

Use Computational Thinking methods to answer the question: *"Who has more sisters – boys or girls?"* 

Thoughts, results and analysis		



### **Session 4: Teaching & Learning Programming**

### Learning Challenges faced by Novice Programmers

Use the space below to take notes from the slideshow presentation.

Notes		

Use the space provided on the next page to record your personal account of learning how to program

You may find the following questions useful to guide you:

- What was the first programming language you learned?
- What did your first program do?
- If you ever learned a second programming language, how did the learning experience differ the second time around?
- Did you ever experience an Aaahhh!! moment when learning programming?
- Were there any programming constructs you found particularly difficult/easy to grasp?
- What was the balance between theory and practical?
- What were the practicals like?
- What approach was taken by your teachers?
- In what ways might Computer Science differ from other subjects in terms of learning (later we can ask the same question in terms of teaching)



How did you learn how to program?

### What were the main challenges for you?



### **Programming Pedagogies: Warmup Activity**





### **Question 3**

return a + b + y

print(calculate(x+1, y))

x = 1

y = x + 1

### PDST

Given x = 20 and y = 9 which of the following Python statements does NOT output the integer 2 exactly?



Notes	
Question 4 What output does the Python code below display?	
def calculate(y, x): a = x b = y + 1 A. 4 B. 5	<ul> <li>Well done – this is the final</li> </ul>

C. 6

D. 7

E. 8

question!

Almost there -

Notes



### **Programming Pedagogies: Peer Instruction**

The warmup activity is an example of a slightly modified version of a programming pedagogy called *peer instruction*.

Peer instruction is a well-evidenced pedagogical strategy developed by Eric Mazur at Harvard University and has been used successfully in Physics, Mathematics and Computer Science. It involves a combination of flipped learning and collaborative working - based on carefully designed Multiple Choice Questions (MCQs) with structured discussion and voting.

Peer instruction offers a way of assessing whether novices really understand concepts that require a precise understanding. It is especially useful for testing *faulty mental models*.

The five key stages of peer instruction are illustrated in the diagram below



Research findings show that when used as an alternative to teacher explanation peer instruction<sup>1</sup>

- has a statistically significant effect on learning
- is twice as effective as a good teacher explanation
- develops a better sense of self efficacy especially among girls

For more information on peer instruction see <u>http://peerinstruction4cs.org</u>

<sup>&</sup>lt;sup>1</sup> Source: Strategies for teaching programming (Sue Sentence, CAS South East Conference, July 2017)



### **Programming Pedagogies: Parson's Problem**

Example 1: Students are asked to arrange blocks of code to perform a specified task.

### Example 1: Parson's Problem



Arrange the blocks of code below into the correct order



The final program should generates a random number, prompts the user to enter a guess and display a message telling the user if the guess was correct, too low or too high.

The program should always display the string Goodbye at the end.

Example 2: Students are asked to arrange blocks of code to perform a specified task but this time with extra lines of code which are not needed (called *distractors* or red herrings).

### Example 2: Parson's Problem



Re-arrange the jumbled up lines shown below so that the program prompts the end-user to enter two integers and then computes and displays their sum.

number2 = int(number2)
number1 = int(input("Enter first number: "))
 sum = sum + number1
 number1 = int(number1)
 print(number1, "+", number2, "=", sum)
 number2 = input("Enter second number: ")
 print("The answer is sum")
 sum = number1 + number2

Warning! There are three extra lines that you won't need.



### **Programming Pedagogies: PRIMM**

Students are ...

- > asked to **Predict** what code will do
- > Run it and see if their prediction was correct
- > given some tasks to help them to Investigate the code
- > asked to **Modify** the code to do make it do different things
- > asked to **Make** a new program

### **PRIMM Example**

```
import random
1.
2.
3. number = random.randint(1, 10)
#print(number)
5.
6. guess = int(input("Enter a number between 1 and 10: "))
7.
8. if guess == number:
print("Your guess was correct")
     print("Goodbye")
10.
11.else:
print("Incorrect guess")
13.
     print("Goodbye")
```

**Predict**: Discuss in pairs. What do you think the above program will do? Be precise.

**Run**: Download the program / key it in. Execute the program. Test your prediction. Were you correct?





### Investigate:

- 1. Uncomment line 4. What happens?
- 2. What is the purpose of line 4?
- 3. What would happen if you removed int from line 6?
- 4. Try changing == to != on line 8. What happens?
- 5. What if == was changed to = ?
- 6. What would happen if you don't enter an integer?
- 7. Try removing a bracket (anywhere). What happens?
- 8. Annotate each line of the program

### Modify:

- 1. Change the program so that it generates a number between 1 and 100
- Change the program so that there is only one print ("Goodbye") statement (without altering the logic)
- 3. Extend the program so that it tells the user if the number entered was too high or too low
- 4. Design an algorithm based on the program that would give the user 3 guesses
- 5. Get the computer to generate 4 numbers (lotto) OR ask the user how many numbers to generate

**Make:** Write a program that generates two random numbers and prompts the user to enter their product. The first number should be a single digit (i.e. 0-9) and the second number should be 2-digits (i.e. 10-99)

Notes		



Leaving Certificate Computer Science National Workshop 1 - Participant Booklet

Additional notes:		







### info@pdst.ie | pdst.ie y f





The PDST is funded by the Teacher Education Section (TES) of the Department of Education and Skills (DES) and is managed by Dublin West Education Centre