codekingdoms

Code Kingdoms
Learning: What, where, when and how

for kids, with kids, by kids.

codekingdoms

Resources overview

We have produced a number of resources designed to help people use Code Kingdoms. There are introductory guides to all parts of the product and classroom materials to help teach lessons around Code Kingdoms.

codekingdoms

Code Kingdoms
Learning: What, where, when and how

for kids, with kids, by kids.

Code Kingdoms Learning: What, where, when and how

A summary of the Code Kingdoms approach to learning.

codekingdoms

Code Kingdoms
Teacher Guide

for kids, with kids, by kids.

Teacher Guide

An overview for teachers. Describes the Code Kingdoms learning ethos and details the different parts of the product.

codekingdoms

Code Kingdoms
Dashboard Guide

for kids, with kids, by kids.

Dashboard Guide

A beginner's guide to using our group management tool. Describes everything from registering for an account to assessing the progress of your kids.

codekingdoms

codekingdoms
Code Kingdoms
Sandbox Guide
for kids, with kids, by kids.

Sandbox guide

A guide to using our unstructured creation environment. Learn everything from using the menus to making great puzzles.

codekingdoms
Code Kingdoms
Unit 1 Introducing Code Kingdoms
for kids, with kids, by kids.

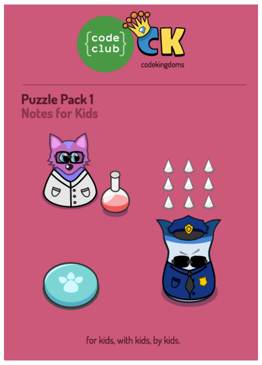
Unit 1: Introducing Code Kingdoms

An introductory unit of six 'off-the-shelf' lesson plans. Targeted at KS2 kids.

codekingdoms
Code Kingdoms
Unit 2 Learning a Language
```
function onPickup() {
    this.health = 12
    this.speak( "happy"
}
```
for kids, with kids, by kids.

Unit 2: Learning a language

Six 'off-the-shelf' lesson plans designed to teach kids the basic of JavaScript


code club
codekingdoms
Puzzle Pack 1
Notes for Kids
for kids, with kids, by kids.

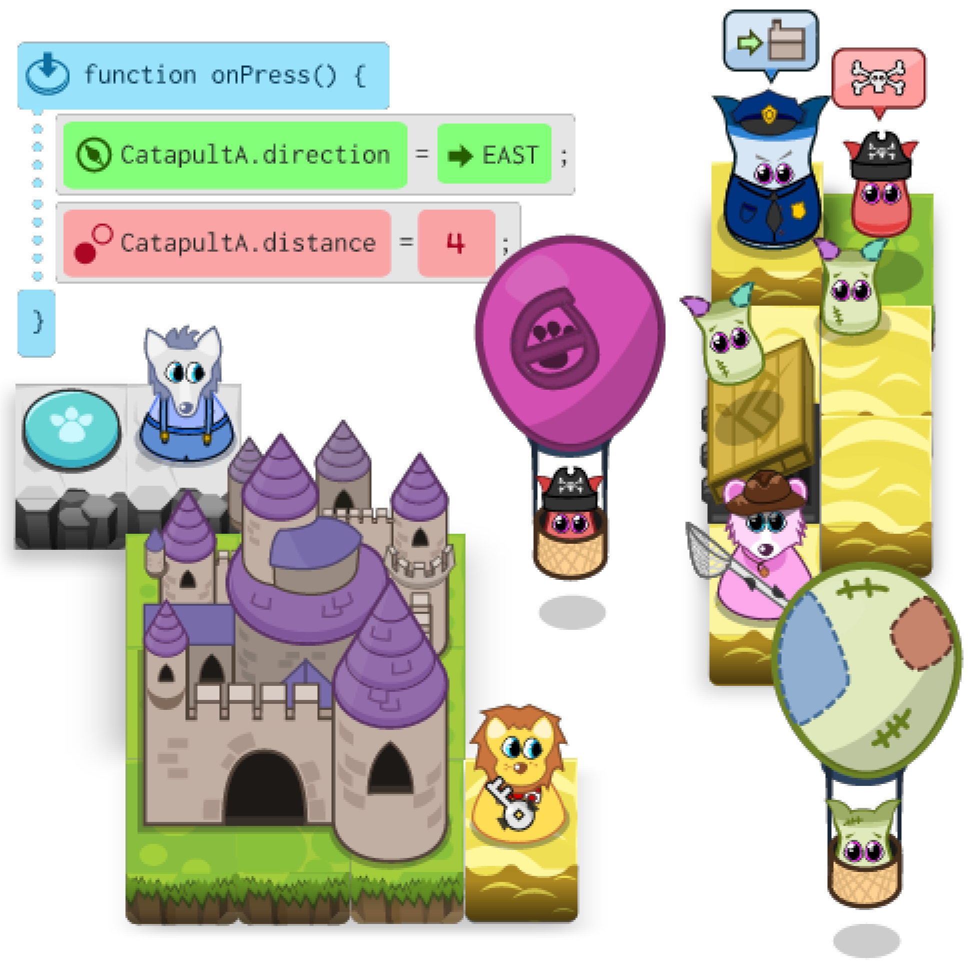
Puzzle Packs

A guide to building specific puzzles in Creative mode. Step-by-step instructions from start to finish. Four puzzles per pack.

Code Kingdoms Learning: What, When, Where and How

What are kids learning?

In Code Kingdoms kids learn a real programming language, JavaScript, alongside Computational Thinking and Computing skills. The tables below detail our full curriculum and specific learning outcomes achieved through play. Our curriculum was based on the popular Computing at School Progression Pathways document.



```
function onPress() {
    CatapultA.direction = EAST ;
    CatapultA.distance = 4 ;
}
```

codekingdoms

JavaScript

JavaScript is one of the most popular programming languages in the world. It is used to build websites, design apps and create games among many other uses. We chose it as the language of CK because it is used to program behaviours of characters and beginners can easily see the effects of their code.

Stage 1 - Sequence & Selection

1.1	Basic JavaScript: I understand how to execute basic JavaScript.
1.2	Constant parameters in functions: I understand how parameters can affect the output of a function.
1.3	Basic functions: I understand how to use JavaScript functions to achieve my aims.
1.4	Changing values - numbers, strings, Booleans:I understand the difference among different types of values and can explain in what situations I might use each.
1.5	Operators: I can use a variable to represent a value that varies in my program and use operators and expressions to change the value.
1.6	Variable parameters in functions: I understand how parameters can affect the output of a function.

Stage 2 - Conditionals and Loops

2.1	Conditional expressions: Using my knowledge of comparison operators, values, and booleans, I can write a conditional expression that results in true or false
2.2	Conditional expressions: I can explain how a conditional expression is used to control the flow of execution in a program
2.3	Conditional expressions for repetition (loops): I can describe how a conditional expression is used in conditional statements and loops
2.4	While loop: I can explain how to use a loop to solve a practical problem.
2.5	For loop: I can explain how to use a loop to solve a practical problem.
2.6	Loop selection - for / while?: I can explain the difference between a while loop and a for loop.

codekingdoms

Stage 3 - Functions

3.1	Grouping statements as functions: I can identify sequences of statements that can be grouped into a function to accomplish a common task.
3.2	Abstraction and functions: I can describe an action that may need to be repeated and create a function to perform that action using statements.
3.3	Returning values from functions: I understand when a function might need to return a value. I can create a function that returns a value, and use that function's value in another part of my code.
3.4	Customising functions: Parameters and arguments: I understand the difference between parameters and arguments and can describe how each are used.
3.5	Customising functions: Different behaviours for different argument values: By using arguments and parameters, I can create a customizable function that performs slightly different behavior depending on the value of the arguments.
3.6	Designing Functions: I can design a simple function and use it to explore how different argument values affect the value that is returned.
3.7	Setting global variables: I can use global variables to share data across different functions

Computational Thinking

Computational thinking incorporates a wide set of skills that include thinking like a computer, solving problems and applying solutions to other situations. It is split into five areas in Code Kingdoms: algorithmic thinking, evaluation, decomposition, abstraction and generalisation.

Algorithmic Thinking

A1	Writing instructions that if followed in a given order (sequences) achieve a desired effect
A2	Writing instructions that use arithmetic and logical operations to achieve a desired effect
A3	Writing instructions that store, move and manipulate data to achieve a desired effect; (variables and assignment)
A4	Writing instructions that choose between different constituent instructions (selection) to achieve a desired effect;
A5	Writing instructions that repeat groups of constituent instructions (loops/iteration) to achieve a desired effect;
A6	Grouping and naming a collection of instructions that do a well-defined task to make a new instruction (subroutines, procedures, functions, methods);
A7	Writing instructions that involve subroutines use copies of themselves to achieve a desired effect (recursion);
A8	Writing sets of instructions that can be followed at the same time by different agents (computers or people) to achieve a desired effect (Parallel thinking and processing, concurrency);
A9	Writing a set of rules to achieve a desired effect (declarative languages);
A10	Using a standard notation to represent each of the above;
A11	Creating algorithms to test a hypothesis;
A12	Creating algorithms that give good, though not always the best, solutions (heuristics);
A13	Creating algorithmic descriptions of real world processes so as to better understand them (computational modelling);
A14	Designing algorithmic solutions that take into account the abilities, limitations and desires of the people who will use them

Evaluation

E1	Assessing that an algorithm is fit for purpose;
E2	Assessing whether an algorithm does the right thing (functional correctness);
E3	Designing and running test plans and interpreting the results (testing);
E4	Assessment whether the performance of an algorithm is good enough;
E5	Comparing the performance of algorithms that do the same thing;
E6	Making trade-offs between conflicting demands;
E7	Assessment of whether a system is easy for people to use (usability);
E8	Assessment of whether a system gives an appropriately positive experience when used (user experience);
E9	Assessment of any of the above against set criteria;
E10	Stepping through algorithms/code step by step to work out what they do (dry run / tracing);
E11	Using rigorous argument to justify that an algorithm works (proof);
E12	Using rigorous argument to check the usability or performance of an algorithm (analytical evaluation);
E13	Using methods involving observing a system in use to assess its usability or performance (empirical evaluation);
E14	Judging when an algorithmic solution is good enough even if it is not perfect;

Decomposition

D1	Breaking down artefacts (whether objects, problems, processes, solutions, systems or abstractions) into constituent parts to make them easier to work with;
D2	Breaking down a problem into simpler but otherwise identical versions of the same problem that can be solved in the same way (Recursive and Divide and conquer strategies)

Abstraction

Ab1	Reducing complexity by removing unnecessary detail;
Ab2	Choosing a way to represent artefacts (whether objects, problems, processes or systems) to allow it to be manipulated in useful ways;
Ab3	Hiding the full complexity of an artefact, whether objects, problems, processes, solutions, systems (hiding functional complexity);
Ab4	Hiding complexity in data, for example by using data structures;
Ab5	Identifying relationships between abstractions;
Ab6	Filtering information when developing solutions;

Generalisation

G1	Identifying patterns and commonalities in problems, processes, solutions, or data.
G2	Adapting solutions or parts of solutions so they apply to a whole class of similar problems;
G3	Transferring ideas and solutions from one problem area to another

Computing

Alongside JavaScript and Computational Thinking, kids also learn general computing skills. This skill set is comprised of Algorithms and Programming & Development and complements the other two strands of learning, whilst being closely aligned with the National Curriculum.

Concept difficulty progresses through coloured phases: pink. yellow, orange and blue.

Algorithms

PA1	I know what an algorithm is and I can express simple algorithms using symbols.
PA2	I know that computers need precise instructions.
PA3	I can show care and precision to avoid errors

YA1	I know that algorithms are implemented on digital devices as programs.
YA2	I can design simple algorithms using loops, and selection i.e. if statements.
YA3	I can use logical reasoning to predict outcomes.
YA4	I can find and correct errors i.e. debugging, in algorithms.

OA1	I can designs solutions (algorithms) that use repetition and two-way selection i.e. if, then and else.
OA2	I can use diagrams to express solutions.
OA3	I can use logical reasoning to predict outputs, showing an awareness of inputs.

BA1	I can show an awareness of tasks best completed by humans or computers.
BA2	I can designs solutions by decomposing a problem and creates a sub-solution for each of these parts (decomposition).
BA3	I know that different solutions exist for the same problem.

PuA1	I know that iteration is the repetition of a process such as a loop.
PuA2	I know that different algorithms exist for the same problem.
PuA3	I can represent solutions using a structured notation.
PuA4	I can identify similarities and differences in situations and can use these to solve problems.

Programming & Development

PP1	I know that users can write their own programs.
PP2	I can create a simple program.
PP3	I can run, check and change programs.
PP4	I know that programs run by following precise instructions.

YP1	I can use arithmetic operators, if statements, and loops, within programs.
YP2	I can use logical reasoning to predict the behaviour of programs.
YP3	I can find and correct simple semantic errors i.e. debugging, in programs.

OP1	I can create programs that implement algorithms to achieve given goals.
OP2	I can declare and assign variables.
OP3	I can use post-tested loops e.g. 'until', and a sequence of selection statements in programs, including an if, then and else statement.

BP1	I know the difference between, and can appropriately use if and if, then and else statements.
BP2	I can use a variable and relational operators within a loop to govern termination.
BP3	I can design, write and debug modular programs using procedures.
BP4	I know that a procedure can be used to hide the detail with sub-solution (procedural abstraction).

PuP1	I know that programming bridges the gap between algorithmic solutions and computers.
PuP2	I have practical experience of a high-level textual language, including using standard libraries when programming.
PuP3	I can use a range of operators and expressions e.g. Boolean, and applies them in the context of program control.
PuP4	I can select the appropriate data types.

When and where are kids learning?

In CK Home, kids learn through experiencing increasingly difficult concepts as their level increases through the game.

More specifically, they develop different skills in different areas of the game:

- Exploring new lands - players solve puzzles using Computational thinking
- Building puzzles to protect their land - computing and JavaScript are needed to code these increasingly complex puzzles

In CK School, kids learn the same skills through building and solving puzzles, but teachers can also set specific puzzles to assess specific learning outcomes. In addition to this, they can set Sandbox mode where kids have the opportunity for open-ended learning because they have the freedom to create the puzzles they want to code.

How are kids learning?

Code Kingdoms develops a range of skills through play. They learn through experience rather than being taught by a course. The platform ensures that each type of learning activity allows kids to learn in a different way and develop different skills.

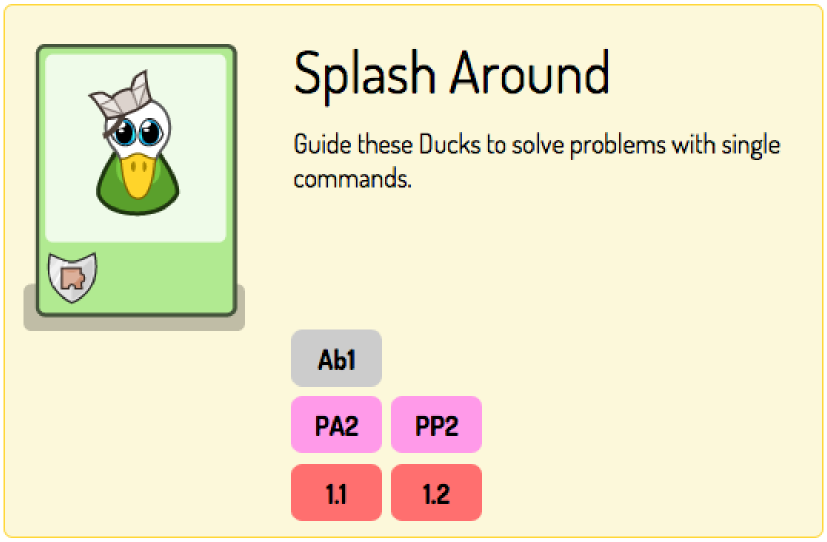
Completing Adventures / exploring lands	Kids learn through problem-solving and experimentation.
Building Puzzles to defend lands	Kids learn through guided but independent learning with additional elements of game design and trial and error.
Sandbox	Kids learn by being creators. They have the freedom to create their own lands and code everything within it!

Examples of Learning Activities

Level 1 Adventure: Splash Around

By completing this adventure kids will learn:

JavaScript	Basic JavaScript: I understand how to execute basic JavaScript.
	Constant parameters in functions: I understand how parameters can affect the output of a function.
Computational Thinking	Ab1: Reducing complexity by removing unnecessary detail.
Computing Progression	PA2: I know that computers need precise instructions. PP2: I can create a simple program.



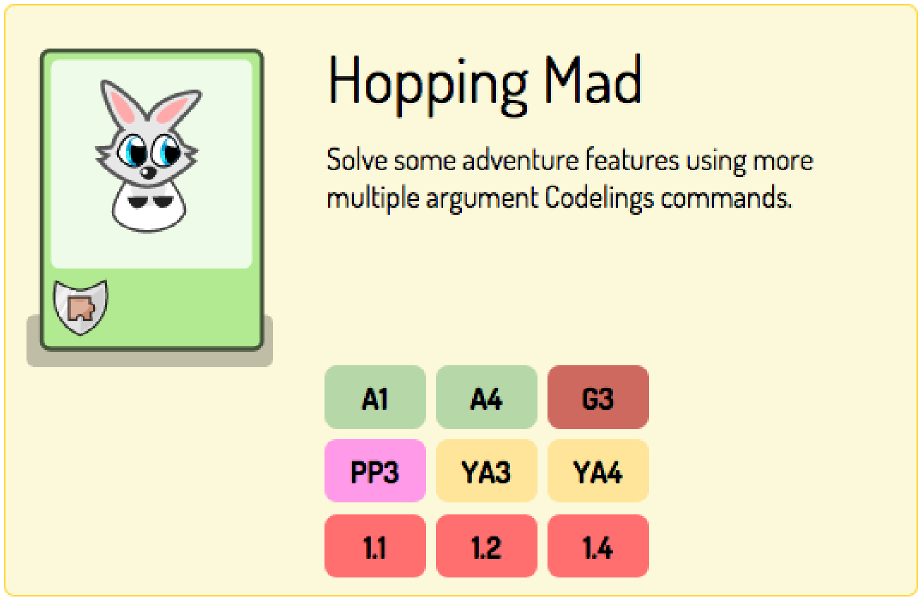
Splash Around

Guide these Ducks to solve problems with single commands.

Level 4 Adventure: Hopping Mad

By completing this adventure kids will learn:

JavaScript	Basic JavaScript: I understand how to execute basic JavaScript.
	Constant parameters in functions: I understand how parameters can affect the output of a function.
	Changing values - numbers, strings, Booleans:I understand the difference among different types of values and can explain in what situations I might use each.
Computational Thinking	A1: Writing instructions that if followed in a given order (sequences) achieve a desired effect. A4: Writing instructions that choose between different constituent instructions (selection) to achieve a desired effect.
	G3: Transferring ideas and solutions from one problem area to another
Computing Progression	PP3: I can run check and change programs. YA3: I can use logical reasoning to predict outcomes. YA4: I can find and correct errors i.e. debugging, in algorithms.



Hopping Mad

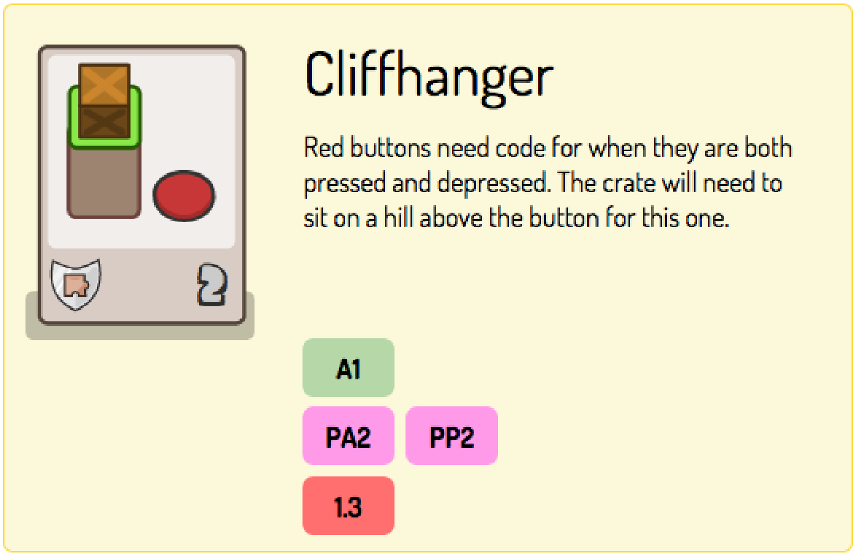
Solve some adventure features using more multiple argument Codelings commands.

A1	A4	G3
PP3	YA3	YA4
1.1	1.2	1.4

Level 2 Puzzle: Cliffhanger

By building this puzzle kids will learn:

JavaScript	Basic functions: I understand how to use JavaScript functions to achieve my aims.
Computational Thinking	A1: Writing instructions that if followed in a given order (sequences) achieve a desired effect.
Computing Progression	PA2: I know that computers need precise instructions. PP2: I can create a simple program.

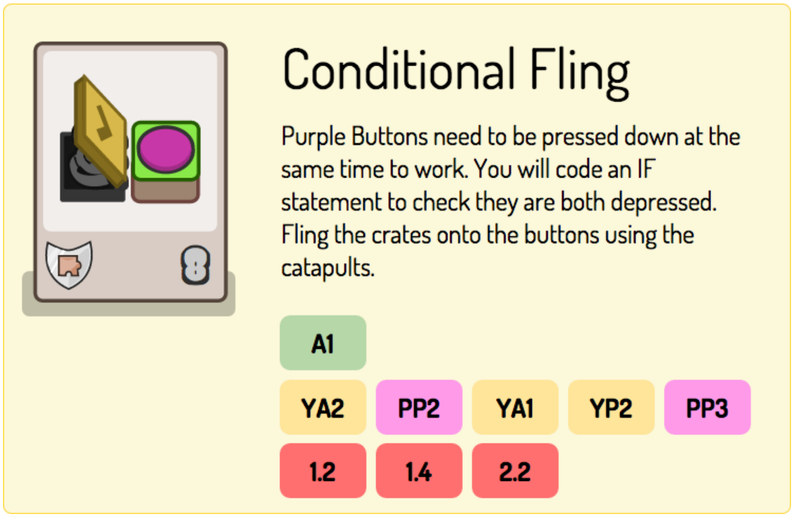

Cliffhanger

Red buttons need code for when they are both pressed and depressed. The crate will need to sit on a hill above the button for this one.

Level 8 Puzzle: Conditional Fling

By building this puzzle kids will learn:

JavaScript	Changing values (numbers, strings, Booleans): I understand the difference among different types of values and can explain in what situations I might use each. Constant parameters in functions:I understand how parameters can affect the output of a function. Conditional expressions: Using my knowledge of comparison operators, values, and booleans, I can write a conditional expression that results in true or false.
Computational Thinking	A1: Writing instructions that if followed in a given order (sequences) achieve a desired effect.
Computing Progression	PP2: I can create a simple program. PP3: I can run check and change programs. YA1: I know that algorithms are implemented on digital devices as programs. YA2: I can design simple algorithms using loops, and selection i.e. if statements. YP2: I can use logical reasoning to predict the behaviour of programs.



Conditional Fling

Purple Buttons need to be pressed down at the same time to work. You will code an IF statement to check they are both depressed. Fling the crates onto the buttons using the catapults.

A1

YA2 PP2 YA1 YP2 PP3

1.2 1.4 2.2