



The Digital Skills Standard

ICDL Digital Student COMPUTING

Syllabus 1.0



Learning Material

Provided by:

PeopleCert Education
for Coding Bootcamp Students

Copyright ICDL Foundation 2015 - 2019. All rights reserved. Reproducing, repurposing, or distributing this courseware without the permission of ICDL Foundation is prohibited.

ICDL Foundation, ICDL Europe, ICDL, ECDL and related logos are registered business names and/or trademarks of ECDL Foundation.

This courseware may be used to assist candidates to prepare for the ICDL Foundation Certification Programme as titled on the courseware. ICDL Foundation does not warrant that the use of this courseware publication will ensure passing of the tests for that ICDL Foundation Certification Programme.

The material contained in this courseware does not guarantee that candidates will pass the test for the ICDL Foundation Certification Programme. Any and all assessment items and / or performance-based exercises contained in this courseware relate solely to this publication and do not constitute or imply certification by ICDL Foundation in respect of the ECDL Foundation Certification Programme or any other ICDL Foundation test. This material does not constitute certification and does not lead to certification through any other process than official ICDL Foundation certification testing.

Candidates using this courseware must be registered with the National Operator before undertaking a test for an ICDL Foundation Certification Programme. Without a valid registration, the test(s) cannot be undertaken and no certificate, nor any other form of recognition, can be given to a candidate. Registration should be undertaken at an Approved Test Centre.

JavaScript is a registered trademark of the JavaScript Software Foundation. JavaScript and its standard libraries are distributed under the JavaScript License. Details are correct as of December 2016. Online tools and resources are subject to frequent update and change.

ICDL Computing

With the increased use of computers in all areas of life, there is a growing interest in learning about the fundamentals of computing, including the ability to use computational thinking and coding to create computer programs.

The ICDL Computing module sets out the skills and competences relating to computational thinking and coding and will guide you through the process of problem solving and creating simple computer programs. Based on the ICDL Computing syllabus, this module will help you understand how to use computational thinking techniques to identify, analyse and solve problems, as well as how to design, write and test simple computer programs using well structured, efficient and accurate code.

On completion of this module you will be able to:

- Understand key concepts relating to computing and the typical activities involved in creating a program.
- Understand and use computational thinking techniques like problem decomposition, pattern recognition, abstraction and algorithms to analyse a problem and develop solutions.
- Write, test and modify algorithms for a program using flowcharts and pseudocode.
- Understand key principles and terms associated with coding and the importance of well-structured and documented code.
- Understand and use programming constructs like variables, data types, and logic in a program.
- Improve efficiency and functionality by using iteration, conditional statements, procedures and functions, as well as events and commands in a program.
- Test and debug a program and ensure it meets requirements before release.

What are the benefits of this module?

ICDL Computing has been developed with input from computer users, subject matter experts, and practising computing professionals from all over the world to ensure the relevance and range of module content. It is useful for anyone interested in developing generic problem solving skills and it also provides fundamental concepts and skills needed by anyone interested in developing specialised IT skills.

Once you have developed the skills and knowledge set out in this book, you will be in a position to become certified in an international standard in this area – ICDL Computing.

For details of the specific areas of the ICDL Computing syllabus covered in each section of this book, refer to the ICDL Computing syllabus map at the end of the learning materials book.

ICDL COMPUTING

LESSON 1 – THINKING LIKE A PROGRAMMER	1
1.1 Computational Thinking	2
1.2 Instructing a Computer	9
1.3 Review Exercise	12
LESSON 2 – SOFTWARE DEVELOPMENT	14
2.1 Precision of Language	15
2.2 Computer Languages	16
2.3 Text About Code	18
2.4 Stages in Developing a Program	20
2.5 Review Exercise	22
LESSON 3 – ALGORITHMS	23
3.1 Steps in an Algorithm	24
3.2 Methods to Represent a Problem	25
3.3 Flowcharts	27
3.4 Pseudocode	31
3.5 Fixing Algorithms	32
3.6 Review Exercise	34
LESSON 4 - GETTING STARTED	36
4.1 Introducing JavaScript	37
4.2 Exploring JavaScript	37
4.3 Saving a Program	39
4.4 Review Exercise	43
LESSON 5 - PERFORMING CALCULATIONS	44
5.1 Performing Calculations with JavaScript	45
5.2 Precedence of Operators	46
5.3 Review Exercise	49
LESSON 6 – DATA TYPES AND VARIABLES	50
6.1 Data Types	51
6.2 Variables	52
6.3 Beyond Numbers	55
6.4 Review Exercise	59

LESSON 7 – TRUE OR FALSE	61
7.1 Boolean Expressions	62
7.2 Comparison Operators.....	63
7.3 Boolean Operators.....	65
7.4 Booleans and Variables	67
7.5 Putting It All Together	70
7.6 Review Exercise	72
LESSON 8 – ARRAY DATA TYPES	73
8.1 Arrays in JavaScript	74
8.2 Creating Arrays in JavaScript.....	75
8.3 Review Exercise	79
LESSON 9 – ENHANCE YOUR CODE	80
9.1 Readable Code.....	81
9.2 Comments	81
9.3 Organisation of Code	82
9.4 Descriptive Names.....	82
9.5 Review Exercise	85
LESSON 10 – CONDITIONAL STATEMENTS.....	86
10.1 Sequence and Statements.....	87
10.2 IF Statement	87
10.3 IF...ELSE Statement.....	89
10.4 Switch Statement.....	91
10.5 Review Exercise	92
LESSON 11 – FUNCTIONS	94
11.1 Function.....	95
11.2 Functions Syntax and Invocation	95
11.3 Review Exercise	98
LESSON 12 – LOOPS.....	99
12.1 Looping.....	100
12.2 Looping with Variables	103
12.3 Variations on Loops	104
12.4 Review Exercise	107
LESSON 13 – OBJECTS - EVENTS	108
13.1 Math Object	109
13.2 String Object.....	110

13.3 Date Object.....	113
13.4 Events.....	115
13.7 Review Exercise	119
LESSON 14 – RECURSION.....	120
14.1 Recursion.....	121
14.2 Recursion vs. Iteration	123
14.3 Review Exercise	124
LESSON 15 – TESTING AND MODIFICATION	125
15.1 Types of Errors	126
15.2 Finding Errors	127
15.3 Testing and Debugging a Program.....	130
15.4 Improving a Program.....	136
15.5 Review Exercise	139
ICDL SYLLABUS.....	140

LESSON 1 – THINKING LIKE A PROGRAMMER

After completing this lesson, you should be able to:

- Define the term computing
- Define the term computational thinking
- Outline the typical methods used in computational thinking: decomposition, pattern recognition, abstraction, algorithms
- Use problem decomposition to break down data, processes, or a complex problem into smaller parts
- Define the term program
- Understand how algorithms are used in computational thinking
- Identify patterns among small, decomposed problems
- Use abstraction to filter out unnecessary details when analysing a problem

1.1 COMPUTATIONAL THINKING



Concepts

Computing is the performing of calculations or processing of data, especially using a computer system.

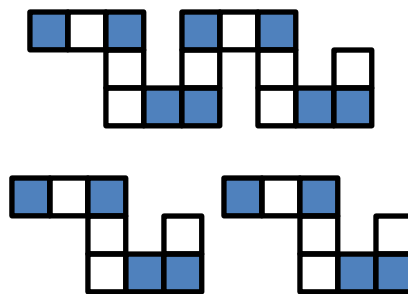
Computational thinking is the process of analysing problems and challenges and identifying possible solutions to help solve them.

Computational thinking is useful in complex problems or tasks such as designing products, cooking, planning events, fixing things that are broken, assembling flat pack furniture, and in many other problem solving situations.

Computational thinking uses four key problem-solving techniques. They can be used in any order and in any combination.

Pattern Recognition

A pattern is a repeated design or feature, like the chorus of a song or a motif on fabric or wallpaper. Patterns can also be found in activities, for example, several different recipes may involve setting the oven to a certain temperature and waiting for it to heat up. This is known as a shared pattern. Pattern recognition involves finding patterns or repetition within complex problems or among smaller related problems.



In the figure above the first set of coloured blocks has been split in two to highlight the presence of a pattern that is being repeated.

Abstraction

Abstraction is the process of extracting the most important or defining features from a problem or challenge. The extracted features provide the information to begin examining the challenge and find potential solutions. It involves filtering out unnecessary details and only looking at information that is relevant to solving the problem. In the task of baking biscuits, it is not important whether you are right or left-handed. The information that is relevant to completing the task includes the

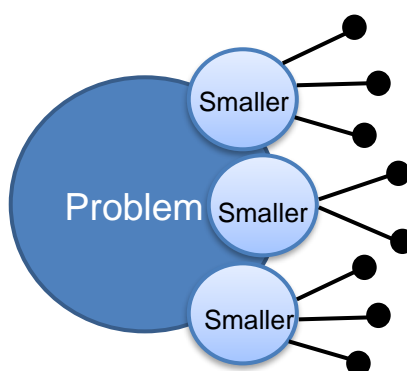
ingredients, the order in which you mix them in, and the duration and temperature of cooking.

In the following example, any unnecessary details have been removed from the second image of the dog so only the information relevant to solving the problem remains – there is a dog and it wants food.



Decomposition

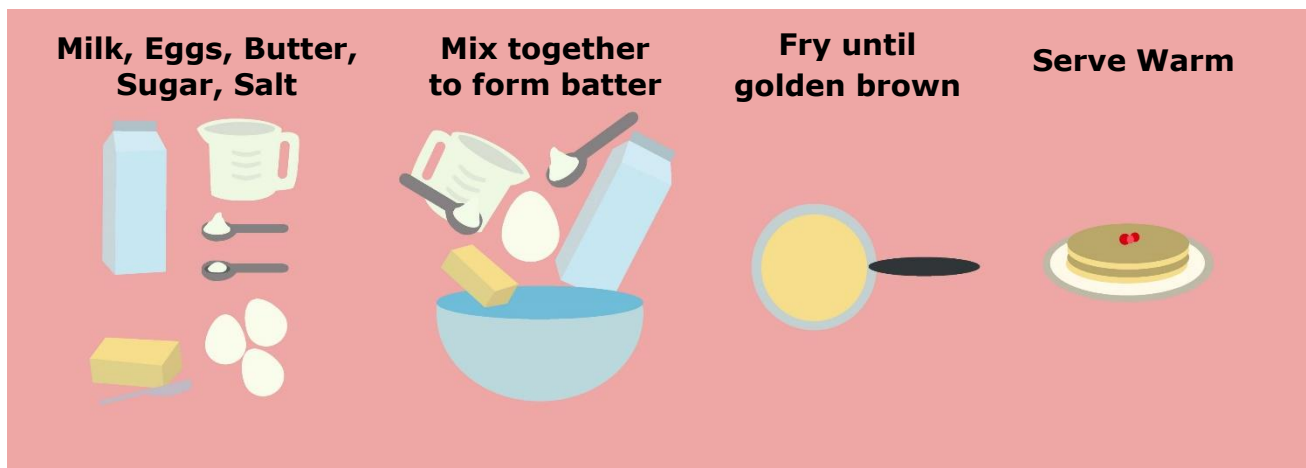
Decomposition involves breaking a complex problem down into smaller, simpler, easier to understand problems. The smaller problems can be broken down into smaller and smaller problems until they are easy to understand and manage. In the task of making biscuits, a smaller problem might be to ensure that the oven is at the right temperature. The image illustrates how a complex problem can be decomposed into smaller and smaller parts or problems.



Algorithms

An algorithm is an organised set of instructions which provides steps for solving a problem or completing a task. For example, an algorithm could be the instructions in a recipe or the calculations for a computer to follow. Algorithmic design is the process of creating these well-defined instructions in the form of steps to solve

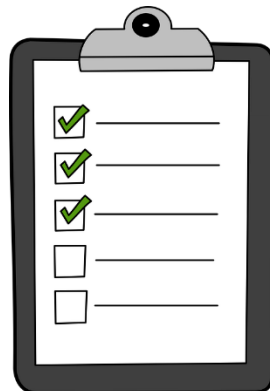
problems or complete tasks successfully. A possible algorithm for making pancakes involves the following steps:



Two additional approaches often included as part of computational thinking are evaluation and generalisation:

Evaluation

Evaluation involves validating the design, of a product or an algorithm, by checking that it does what it's meant to, or solves the problem.



Generalisation

Generalisation is taking a solution and finding a way to make it useful in a wider set of circumstances. For example, you might use symbols instead of words on a products' controls so it can be used regardless of the language spoken. In the following image the basic structure of the flower is always the same and can be used repeatedly but the characteristics such as the colour and shape can be varied.



These six computational thinking skills work together to solve complex problems in computing and beyond.



Steps

Example: Designing a Washing Machine

In this example the techniques of computational thinking are applied to the complex problem of designing a washing machine.



Abstraction

The first step in designing a washing machine is to determine the goal – for example it should be able to wash vigorously or gently, at a low or high temperature and produce clean clothes. The design phase involves listing the features to achieve this goal. Abstraction can be used to help the designer filter out what is relevant and what is not when listing the features to include and exclude. Details that aren't relevant might include things such as the colour of the washing machine or whether the load to be washed has six pairs of socks or three pairs of socks.

Details that are relevant are those that affect the overall functionality and they might include things such as water temperature and whether a programme is for delicate or hard-wearing fabrics. These details are relevant to solving the problem of producing clean clothes without damaging them.

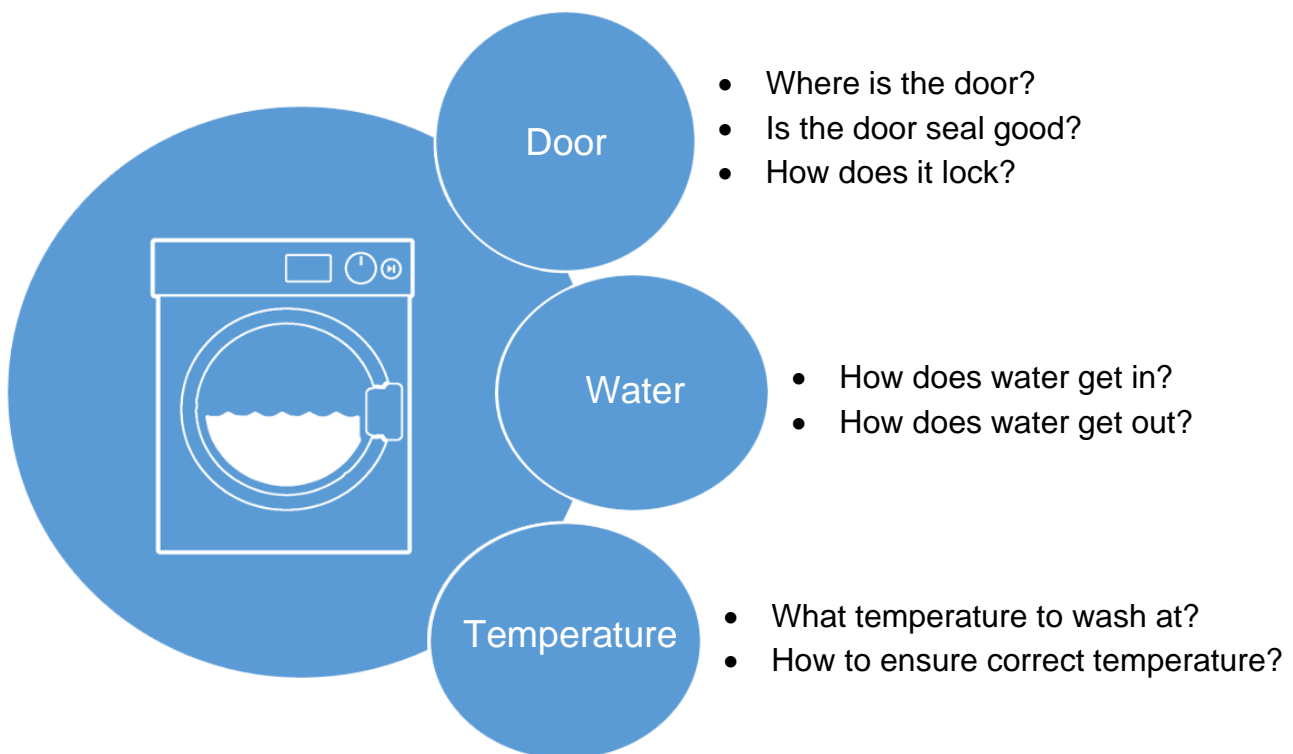
Decomposition

Decomposition can be used to break the problem down into smaller, more manageable problems, such as:

- How do we get clothes in and out of the washing machine?
- How do we get water in and out of the washing machine?
- How do we make sure the water is at the right temperature?

In turn these small problems may be decomposed into even smaller ones:

- Where is a good place for a door on a washing machine?
- How can we make a door that seals properly, so water does not leak out?



Decomposition of the problem of designing a washing machine

Algorithms

Algorithms can be designed to specify the precise steps that the machine should follow for different wash cycles. And there are many other algorithms needed. For example an algorithm can be designed to specify the sequence of precise steps to be followed to manufacture the washing machine.

Evaluation

Designing a product is a big challenge, so early designs will be continuously evaluated. Through evaluation, the designers learn where and how the design can

be improved. For example, if while testing a machine, water leaks out, the design can be modified to prevent this happening in future.

Generalisation

The washing machine design may also be improved by making it universal, for example using symbols instead of words on the controls, so that it can be used by people regardless of their language, or by designing it to work on 110 volts as well as 220 volts for use in different countries. If the design is modified in this way, then it has been generalised.

Pattern Recognition

Pattern recognition has an important role and along with the other five techniques it can be used multiple times throughout the different stages of the design process. It can help with abstraction by highlighting the similarities and differences in aspects of a problem. Pattern recognition can also help with generalisation; where aspects of one particular problem can be related to more general problems. Similarly, evaluation of the design helps to generalise it, by evaluating how the design may be applied in different settings.

The six techniques in computational thinking are not necessarily applied in sequence or at only one stage of the design process. They are used in many stages of design and in different sequences.



Steps

Example: Organising a Music Festival

In this example the techniques of computational thinking are applied to the complex problem of organising a music festival.

Abstraction

To begin to plan a music festival you must understand what a music festival is, so the first step could be to list essential things that a music festival should have:

- Musicians
- Venue
- Marketing and publicity
- Tickets
- Staff etc.

There will be other details to consider in organising the festival, which are not essential such as the colour of the tickets and the exact wording on the tickets.

These can get in the way of the initial planning, and can be worked out later. Using **abstraction**, the non-essential details can be removed from the plan for the moment.

Decomposition and Algorithms

The challenges in organising the musical festival can then be broken down into smaller challenges. In practice, they could be delegated to individual people to solve:

- Someone to liaise with the venue where the festival will take place.
- Someone to deal with publicity.
- Someone to work on bookings and tickets.

These smaller challenges can be further broken down into even smaller problems to solve. This is **decomposition** of the problem and an example of applying the methods of computational thinking.

For example the problem of organising the venue could be broken down as follows:

- Where will it be?
- When is it available?
- What are the car parking arrangements?

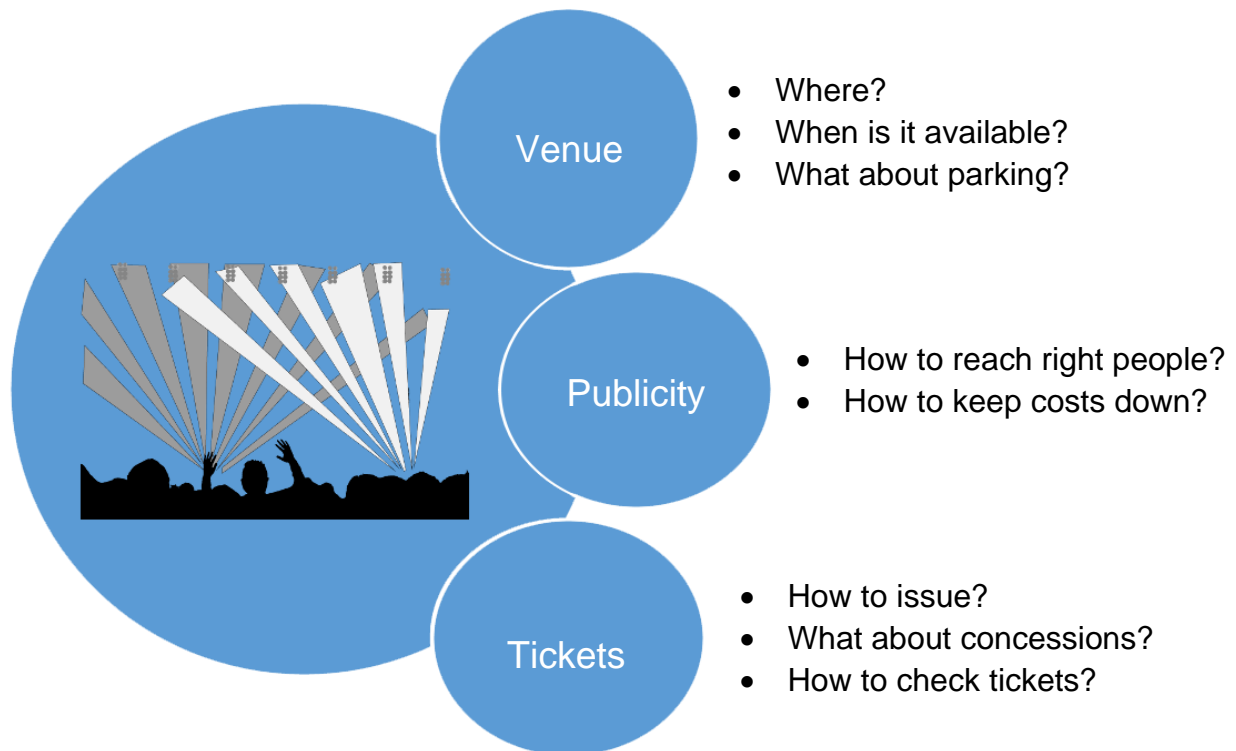
And the smaller problem of the car parking arrangements can be broken down into small processes or **decomposed**. And simple **algorithms** can be created, including some of the following:

- To direct cars to designated spaces.
- To direct cars to the overflow car park when the main one is full.
- To prevent cars blocking important routes.
- To make sure priority parking is kept clear for people who are entitled to it.

The problem of publicity could be broken down as follows:

- How to make the festival interesting so that people will want to go?
- How to advertise to your intended audience event in a cost-effective way?

And the booking and ticketing process can also be **decomposed** further. For example the ticketing process can be broken down into a set of steps, and an **algorithm** created covering design, printing, purchase, delivery, concessions, refunds etc.



Decomposition of the problem of organising a music festival

Evaluation

After the event has run, it can be helpful to gather feedback about what worked well and what did not work so well. This is a way of evaluating the plan. If the festival runs again, the lessons learned from the evaluation will improve the following year's event.

Generalisation

Generalisation is relevant too. A team that has run a music festival might go on to more ambitious goals, for example a series of festivals in a tour, or increasing the duration of the festival, adding more acts, or more kinds of music. The existing solution or formula is reused, but generalised to include new elements.

1.2 INSTRUCTING A COMPUTER



Concepts

Computational thinking is a general problem solving approach but it can also be used as the starting point in creating instructions for computers.

The decomposition of problems into simpler steps leads to development of one or more **algorithms** - collections of well defined, simple steps, to be followed in order to solve a problem. Algorithms can then be presented in ways that computers can understand.



Steps

Example: An algorithm for sorting people by height



In this example you want to sort the people in your class by height. To do this, you could decide on a number of steps to perform:

- Step 1: Line everyone up in a single row.
- Step 2: Decide which end of the row is to be 'tall' and which is to be 'short'.
- Step 3: Repeatedly compare heights and switch students when they are in the wrong place.

This set of steps is an algorithm. However, it is not very detailed. Step 3, for example, could be further decomposed into smaller problems:

- Which end of the row to start at
- How to compare student heights
- What to do if the students are taller or shorter

Once designed correctly, this algorithm for sorting people by height could be modified to solve slightly different problems. A small variation on the algorithm could instead sort people in the class by order of birthday. And with further modifications, it could be used to find out if any two people have the same birthday.



Concepts

Algorithms for Computers

In the algorithm for sorting people by height the instructions are detailed enough for a human to follow, but they aren't detailed enough for a computer to follow. A computer needs to be instructed using a computer language, which is much more detailed and precise than human language.

So for a computer to follow the instructions in the algorithm, the algorithm must be converted into a language that the computer can understand. An algorithm expressed in a form that can be understood and executed by a computer is known as a **program**.

When the computer obeys the steps in a program, it is said to run or execute the program. So a computer program is written in a computer language and the computer works by running or executing the program.

Imagine a robot baking a cake. An instruction in an algorithm to 'put the cake in the oven' would not be detailed enough for the robot (computer) to act on. A robot chef requires a program with much more detailed instructions about how to move its arms and fingers when putting the cake in the oven - and how not to topple over in the process.

1.3 REVIEW EXERCISE

1. Computing is a set of activities that include:
 - a. performing of calculations or processing of data.
 - b. following a recipe to make biscuits
 - c. accurate and careful observation of the stars, over a long period of time.
 - d. accurate and careful observation of chemical reactions.
2. Computational thinking is:
 - a. When computers are solving a difficult problem.
 - b. the process of analysing problems and challenges and identifying possible solutions to help solve them.
 - c. Using a computer to do lots of maths.
 - d. Any activity where a computer and a person work together to do more than either could on their own.
3. Which method is not a typical part of computational thinking?
 - a. Abstraction
 - b. Apprehension
 - c. Decomposition
 - d. Pattern Recognition
4. Which of these is a good example of decomposition of a problem?
 - a. Splitting the task of designing a robot into designing the hand, designing the power source, deciding on and designing the sensors.
 - b. Slicing a cake into 6 equal slices
 - c. Putting an apple into a glass jar and taking photos of it each day to see what happens.
 - d. Using a search engine to find the answer to a question.
5. A program is:
 - a. The detailed rules and regulations of a game or sport
 - b. An algorithm expressed in a form that is suitable for a computer
 - c. The collection of laws and regulations that determine what buildings can be built legally at a particular place.
 - d. A sequence of instructions for a person to follow, e.g. a recipe for baking a cake.
6. Which of these is least likely to be a way that algorithms are used in computational thinking?
 - a. An algorithm may lead to a computer program that when run solves the problem
 - b. An algorithm may lead to a computer program that employs intuition to derive better solutions.
 - c. An algorithm may provide a step by step guide for manufacturing something.
 - d. An algorithm may provide a step by step guide for processes involving people, money and resources.
7. Here are three recipes:

Chocolate cake:

- Set oven to 180°C
- Butter two 9" cake pans
- Mix ingredients with a whisk for one minute
- Transfer mix to cake pans
- Bake for 30 mins
- Allow to cool
- Ice and decorate cake.

Gingerbread men:

- Cream the ingredients together
- Set oven to 190°C
- Roll out the dough to about 1/8" thickness, and cut into gingerbread men shapes.
- Bake until edges are firm, about 10 minutes.

Blueberry muffins:

- Set oven to 185°C
- Grease 18 muffin cups
- Cream butter and sugar until fluffy
- Add other ingredients
- Spoon into muffin cups
- Sprinkle with topping
- Bake for 15 to 20 mins.

Which of these is a pattern common to all three recipes?

- a. Grease 18 muffin cups.
 - b. Ice and decorate.
 - c. Roll out the dough.
 - d. Set oven to some temperature.
8. For designing a program for cooking for a robot chef to use, which of the following would be the most relevant detail?
- a. The colour of the robot.
 - b. What quantities of each ingredient to use.
 - c. Which shop the ingredients were bought from.
 - d. Whether the programmer is right or left handed.

LESSON 2 – SOFTWARE DEVELOPMENT

After completing this lesson, you should be able to:

- Understand the difference between a formal language and a natural language
- Define the term code; distinguish between source code and machine code
- Understand the terms program description, specification
- Recognise typical activities in the creation of a program: analysis, design, programming, testing, enhancement

2.1 PRECISION OF LANGUAGE



Concepts

Types of Language:

Natural Language

Spoken languages like English, French or Chinese are natural languages. A natural language needs context to be clearly understood and avoid ambiguity.

Formal Language

A formal language is strictly structured, with exact and precise rules. It is used in mathematics, chemical equations and computer programs. It is clear and unambiguous.

Computer languages are formal languages. One way formal languages can avoid ambiguity is by using brackets to group words and terms, and by avoiding words like 'it' or 'he' or 'she' where it might not be clear what 'it', 'he' or 'she' refers to.

Here are three different types of brackets used in formal languages. Each type of brackets has a different function, and they can be used to define things like which order a set of steps are expected to be performed in a program.

BRACKET	NAME
()	Round brackets, or parentheses .
{ }	Curly brackets or braces .
[]	Square brackets.

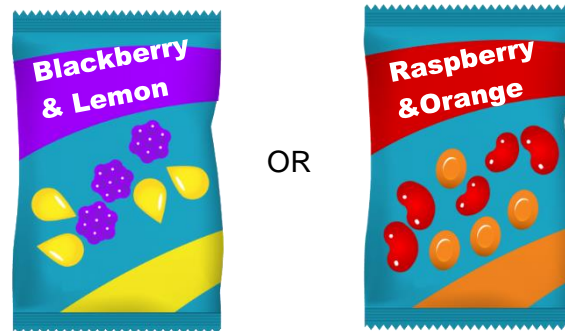


Steps

Example: Ambiguity with 'and' and 'or'

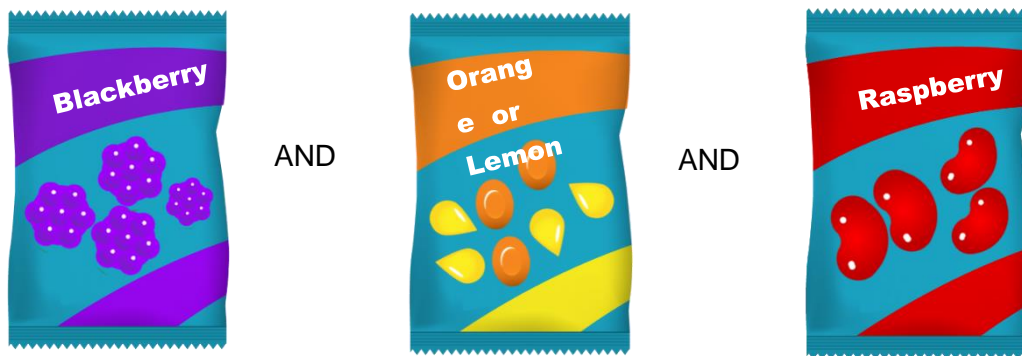
If you ask someone what flavour of sweets they would like, they might say: "Blackberry and Lemon or Raspberry and Orange". You understand the response, but for a computer to understand the sentence, the words need to be grouped correctly.

For the computer, you need to write (Blackberry AND Lemon) OR (Orange AND Raspberry)



(Blackberry AND Lemon) OR (Orange AND Raspberry)

Whereas if you write Blackberry AND (Orange OR Lemon) AND Raspberry, the computer interprets it as:



Blackberry AND (Orange OR Lemon) AND Raspberry

2.2 COMPUTER LANGUAGES



Concepts

Computer languages are designed for writing computer programs that instruct computers to follow a sequence of steps to solve a problem. Computer languages have a smaller vocabulary than natural or spoken languages.

There are many different computer languages. These learning materials use a popular computer language called JavaScript. Some other common languages are Java and C++.



Concepts

Computer Code

The text of a computer program, written as a series of instructions for a computer to execute, is known as code. Different computer languages use different styles of code with different rules and ways of organising instructions, often referred to as syntax.

The work of writing a program is called programming or coding.

People who write programs are called programmers or coders.

There are two kinds of computer code: **source code** and **machine code**.

Source code is the code written by the programmer that humans can understand. It is typed into a computer, usually as text, punctuation and symbols, which contains instructions for the computer. If you learn the formal programming language and its rules (e.g. JavaScript), you can write source code.

Machine code is a series of 1's and 0's created by the computer from the source code that the computers electronic circuits can understand. Creating machine code from source code is known as compiling or interpreting the source code.



Steps

Example: Machine code to lock a door

The instruction to lock a door written in source code might look like this:

Lock(door)

Although this isn't grammatically correct, it does make some sense to people reading it.

A computer reading the source code instructions sees the letters 'L', 'o', 'c', 'k', '(', and so on. The instructions aren't in a format the computer can act on directly. Instead the computer needs the instructions to be converted into a format that it can understand and obey.

The computer has electronic circuits that act on certain patterns or combinations of 1's and 0's. Below is an example of what some patterns of 1's and 0's might mean.

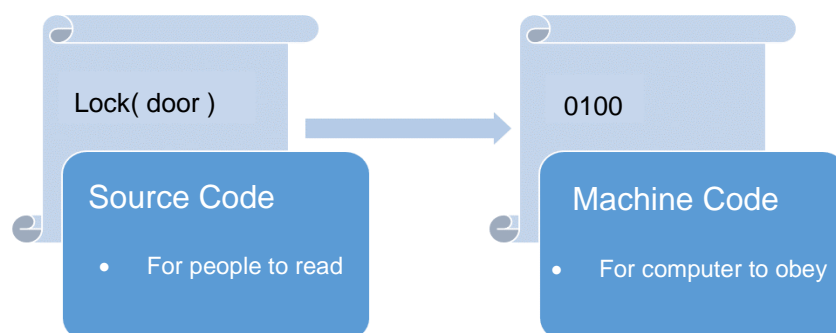
Machine Code	Meaning of the machine code instruction
0001	Sound alarm
0010	Lock windows
0100	Lock door
0110	Lock windows and door

1000	Turn on sprinkler system
1001	Turn on sprinkler system and sound the alarm

Instructions like '0110' are machine code instructions.

In reality the full range of possible machine code instructions for a computer would be much larger. There would, for example, be many instructions for performing arithmetic.

Source code is converted into machine code before a computer obeys the instructions. 'Lock(door)' would be translated to 0100 and the computer could then obey that instruction.



Source Code v Machine Code

In this simple example, one source code instruction 'Lock(door)' corresponds to one machine code instruction, 0100. Usually, a single line of source code requires several machine code instructions one after the other, in the correct sequence.

The earliest computer programmers converted the source code into machine code manually, and created and documented the series of 1's and 0's themselves. The invention of programs to translate source code to machine code made it possible to write larger and more complex computer programs.

2.3 TEXT ABOUT CODE



Concepts

As well as writing source code programmers also need to document their work in the form of text notes. These text notes are not read by the computer, but help the programmer and other colleagues to understand the different stages of the program.

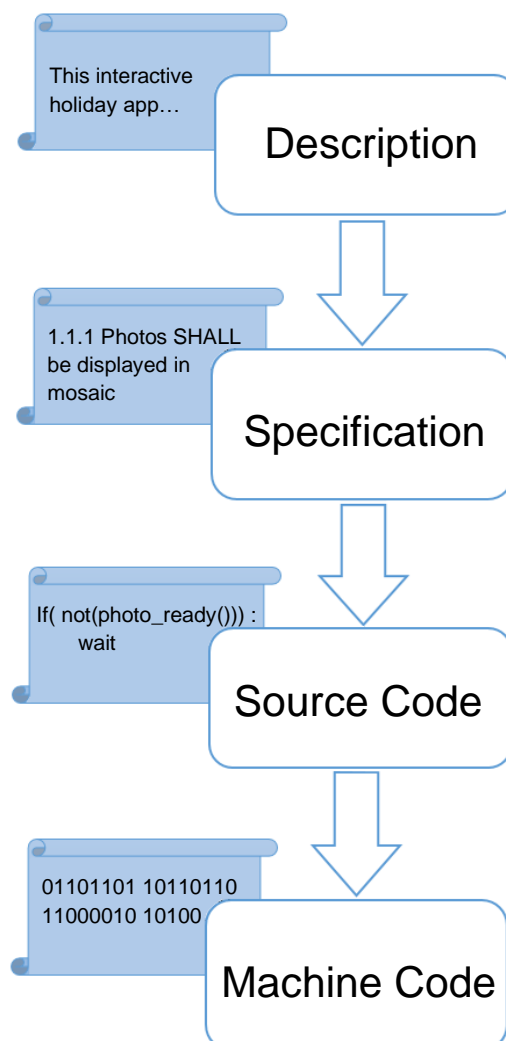
Text Describing a Program

Programmers write program descriptions and specifications to help everyone involved in a project to understand what the program was designed to do and what problem needed to be solved. They can also be used in the evaluation stage to check that the program is working as desired.

A **program description** explains what a program is designed to do and how it works. This could be helpful for other programmers, the user of the end product and also the marketing department to use for promotion and sales.

A **program specification** is the set of requirements that outline what a program will do. Usually, the specification is written before the program. A specification is more detailed than a description, and it states requirements that can be tested.

For example, a specification might include the requirement “The program SHALL dim the screen if the computer has not been used for one minute, to save electricity”. Once the program is written, the program can be tested by leaving the computer running for a minute and checking that the program does indeed dim the screen. The following image describes the possible progression in work on a computer program from development to execution.



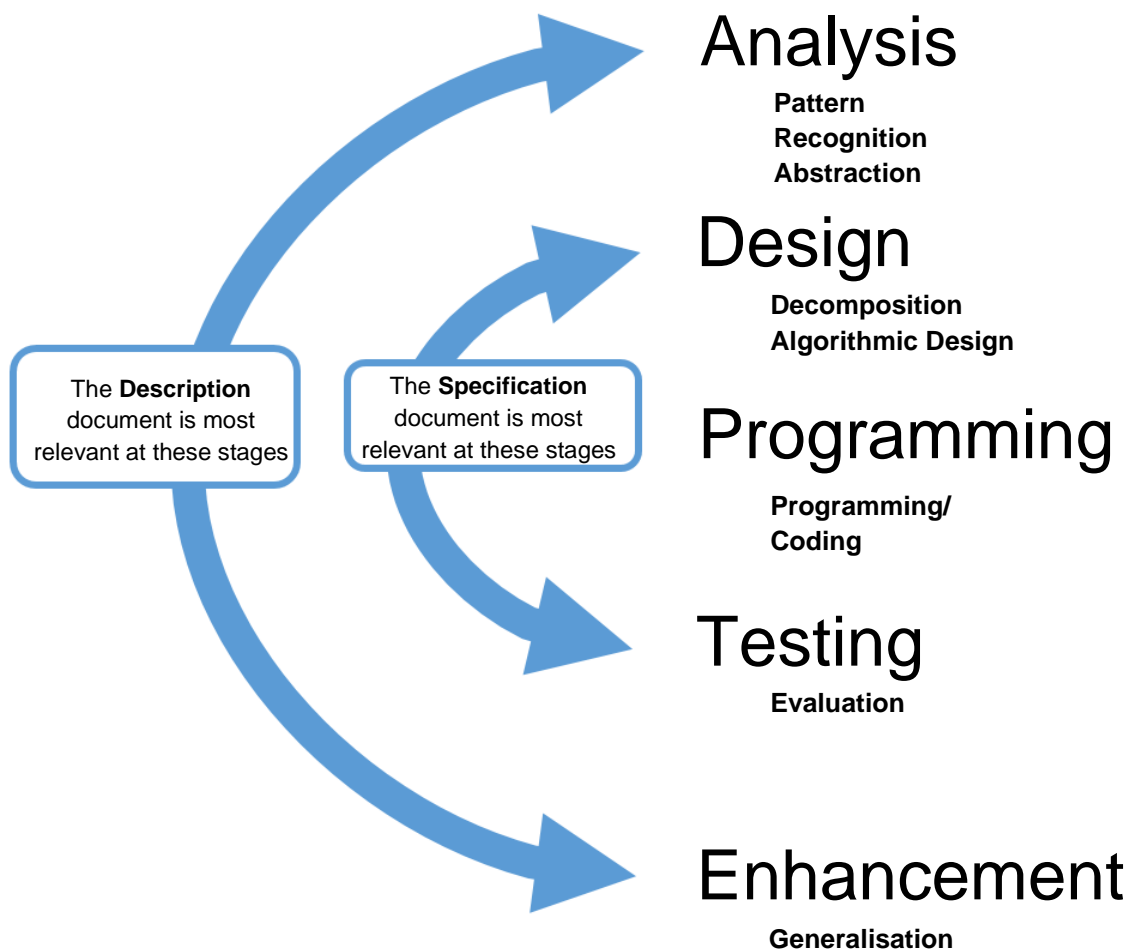
2.4 STAGES IN DEVELOPING A PROGRAM



Concepts

Descriptions and specifications can play an important role in the development of a program.

The following diagram shows some of the main stages and activities in the creation of a program.



Stages in Program Development

Analysis

This involves defining the problems to be solved more clearly. Analysis is largely a process of abstraction, listing all aspects of the problem, identifying the relevant aspects and how they relate to each other.

Design

This involves working out the algorithms (sets of steps to follow) to solve the problem. This uses decomposition to break the problem down into smaller parts, and planning the design of the algorithm.

Programming

This involves writing the program. This includes finding a way to express the algorithm in the chosen computer language.

Testing

This involves checking that the program actually does what was it was intended to. Testing can uncover errors in the logic or syntax of the program.

Enhancement

This involves adding new features to extend the program's capabilities, improve its performance or functionality, or to generalise it for use in different situations.

A general **program description** is typically formulated very early in the project, as part of deciding the scope of what the program will do.

The **program specification** is normally drawn up during the design phase. Some of the requirements in the specification may arise directly from decomposition of the problem into smaller problems. The program specification is examined during testing, as each statement about the program's capabilities must be verified.

In planning enhancements, the program description can be updated to record plans for extending the program's capabilities.

2.5 REVIEW EXERCISE

1. A formal language is:
 - a. A language in which it is not possible to make a mistake.
 - b. A language with defined rules and unambiguous meanings.
 - c. The best language to use when writing a postcard.
 - d. Any language that has the words "and" and "or" in it.
2. English, Arabic and Chinese are:
 - a. Formal languages.
 - b. Computer languages.
 - c. Natural languages.
 - d. Source code.
3. Machine code is.
 - a. Translated into source code so that a computer can obey the instructions.
 - b. A method for secure communication between two computers.
 - c. The 1's and 0's that a computer obeys.
 - d. A written agreement between a programmer and a customer specifying what a program should do.
4. An algorithm written in the JavaScript computer language is an example of:
 - a. Source code
 - b. Machine code
 - c. A program specification
 - d. A program description
5. A program specification is:
 - a. The code that the computer runs.
 - b. The comments in code that a programmer reads.
 - c. The actual electrical signals in a computer, when a program is running.
 - d. A description of what a program should do that is used during designing the program.
6. Match each activity in creating a program, a to e, to their purpose:
 - a) Testing, b) Design, c) Programming, d) Analysis, e) Enhancement

Improving an existing program.	
Clearly defining the problem.	
Checking whether the program works correctly.	
Expressing the algorithm in the chosen computer language.	
Working out an algorithmic approach to solving a problem.	

LESSON 3 – ALGORITHMS

After completing this lesson, you should be able to:

- Define the programming construct term sequence. Outline the purpose of sequencing when designing algorithms
- Recognise some methods for problem representation like: flowcharts, pseudocode
- Recognise flowchart symbols like: start/stop, process, decision, input/output, connector, arrow
- Outline the sequence of operations represented by a flowchart, pseudocode
- Write an accurate algorithm based on a description using a technique like: flowchart, pseudocode
- Fix errors in an algorithm like: missing program element, incorrect sequence, incorrect decision outcome

3.1 STEPS IN AN ALGORITHM



Concepts

Sequencing

Computer algorithms are complex problems broken down into simpler steps or **instructions**. In most algorithms, the instructions are obeyed one after the other in a **sequence**.

A **sequence** is a number of simple instructions that should be carried out one after the other.

In most algorithms the order of the instructions matter. A robot chef would need to put the cake mixture into the tin before putting it in the oven.

Designing sequences of instructions is a crucial skill in programming. A programmer needs to make sure that all the required actions are performed in the right order to accomplish a task or set of tasks. A sequence is the fundamental method of control in a computer program. The sequence is decided upon in the design phase of a project where algorithms and flowcharts are used to create the most efficient and correct sequence of program control.

Input and Output

Instructions often use information from the world outside the computer and do something with it. Computer programs can receive information to work on via **inputs** and give results via **outputs**.

An input

A value from outside the computer that is needed for the instructions to be followed. For example: a measurement of temperature, or a number typed on the keypad of a security/alarm system.

An output

A value calculated, or an action performed, by the computer program, which is displayed to the world outside of it. For example: a light being switched on or off, a security alarm being set, or a message displayed on a screen.



Steps

Example: Inputs and Outputs of a Mobile Phone

A mobile phone takes in information or inputs and processes them and returns results or outputs.

- The touch screen is an important input method, for example when you use it to type a message or tap open an application.
- The screen display is an important output, for example when displaying a list of contacts.

Yes / No Decisions

As you have seen in previous lessons, the order of instructions, or sequence, is important in computing. In general, instructions are followed one after the other, but sometimes there are decisions to be made on what to do next. The algorithm can be designed to ask questions and expect an input to help decide on the next step.

A **yes / no decision** is an instruction that chooses the next instruction to obey. It does this based on whether the answer to a question is yes or no.



Steps

Example: Waiting for an Oven to Heat Up

Before putting a cake mixture into the oven, the oven needs to be at the right temperature. An algorithm could include a yes / no decision to determine this. “Is the oven at the right temperature?” A thermostat will provide the input, yes or no. If the answer is yes, then it is ok to put the cake into the oven. If the answer is no, the algorithm will wait a while and ask the question again.

3.2 METHODS TO REPRESENT A PROBLEM



Concepts

Two techniques to help programmers write programs are pseudocode and flowcharts. These are used to represent the sequence of steps in an algorithm. Algorithms expressed in pseudocode or as flowcharts are rarely expressed precisely enough for a computer to use, but present a series of steps in sufficient detail for a programmer to see how the algorithm is supposed to operate.

Pseudocode:

Pseudocode is an informal way of representing an algorithm, using written instructions in natural language, for example English. The steps in the algorithm are written out as a sequence of instructions. An algorithm written as pseudocode is intended for a human to read rather than a computer.

Pseudocode looks like the code that the program will be written in, but is not quite as precise. If the pseudocode makes sense to read, then the next step is to write the actual code for the program.

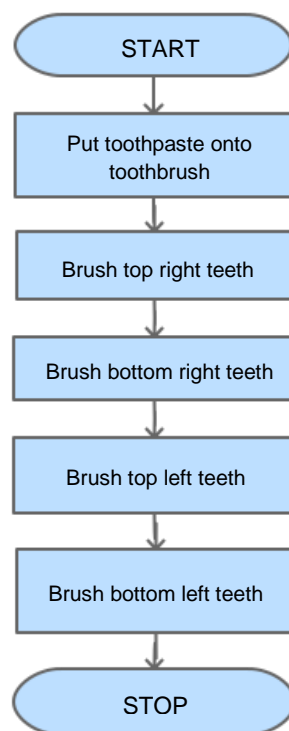
An algorithm for brushing teeth, written as pseudocode could be written as:

```
Put toothpaste onto toothbrush  
Brush top right teeth  
Brush bottom right teeth  
Brush top left teeth  
Brush bottom left teeth  
STOP
```

Flowchart

A flowchart is a pictorial way of representing an algorithm. Flowcharts represent a series of simple steps with arrows showing the progression from step to step. The steps are shown using shaped boxes containing text. A flowchart is a fundamental tool used in the development of computer programs. It allows developers to outline, step by step how they want to solve a problem or how they want a program to behave.

The algorithm for brushing teeth could also be shown as a flowchart:



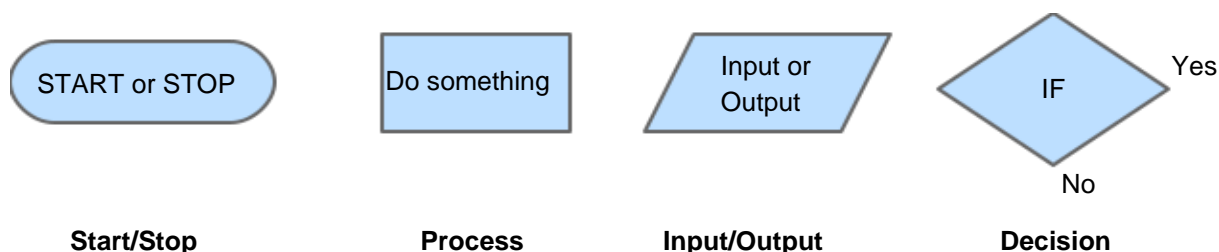
Flowchart for brushing teeth

3.3 FLOWCHARTS



Concepts

Some of the shapes used in a flowchart are shown below:



Flowchart symbols

The shapes represent the following:

Start or Stop

The algorithm starts at the Start box and runs until it reaches a Stop box.

Process

These boxes contain simple instructions to do something, such as add two numbers or look up a word in a dictionary.

Input or Output

These boxes are for interaction with the world outside the computer. For example, an input could be from a movement or temperature sensor. An output could be an action such as turning on or off a light.

Decision

Decision boxes allow for alternative choices in what the algorithm does next. They often have a question in them which will have a yes or no answer. If the answer is 'yes' one route is taken. If the answer is 'no' the other route is taken. This is how algorithms can go beyond simple sequences of steps.

Connecting the Boxes

These are two additional kinds of flowchart symbols:



Arrow



Connector

The arrow and connector are used to connect the flowchart boxes.

Arrow

A directional line drawn to connect two boxes in a flowchart. This arrow shows the direction of flow in an algorithm. These are often referred to as flow lines. For example, instructions in a sequence have arrows between them.

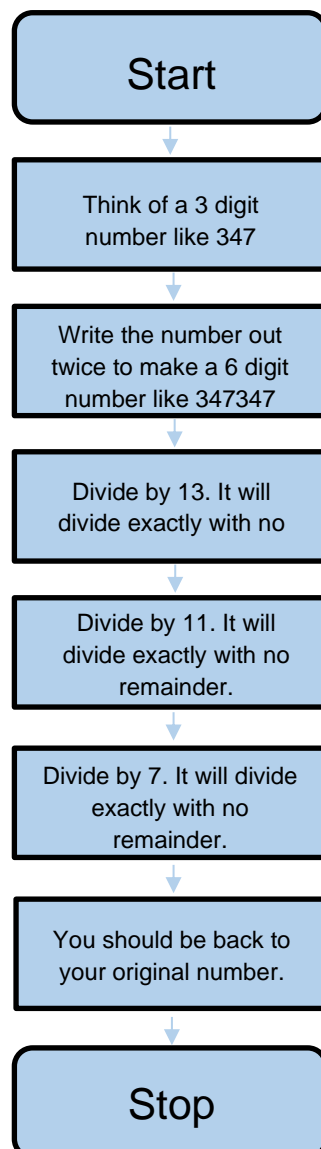
Connector

A small circle in a flowchart that is used to connect two flow lines together. It shows a jump from one point in the process flow to another, for example when answering a “Yes/No” question.



Steps

Example: Flowchart for the 347347 'Magic Trick'



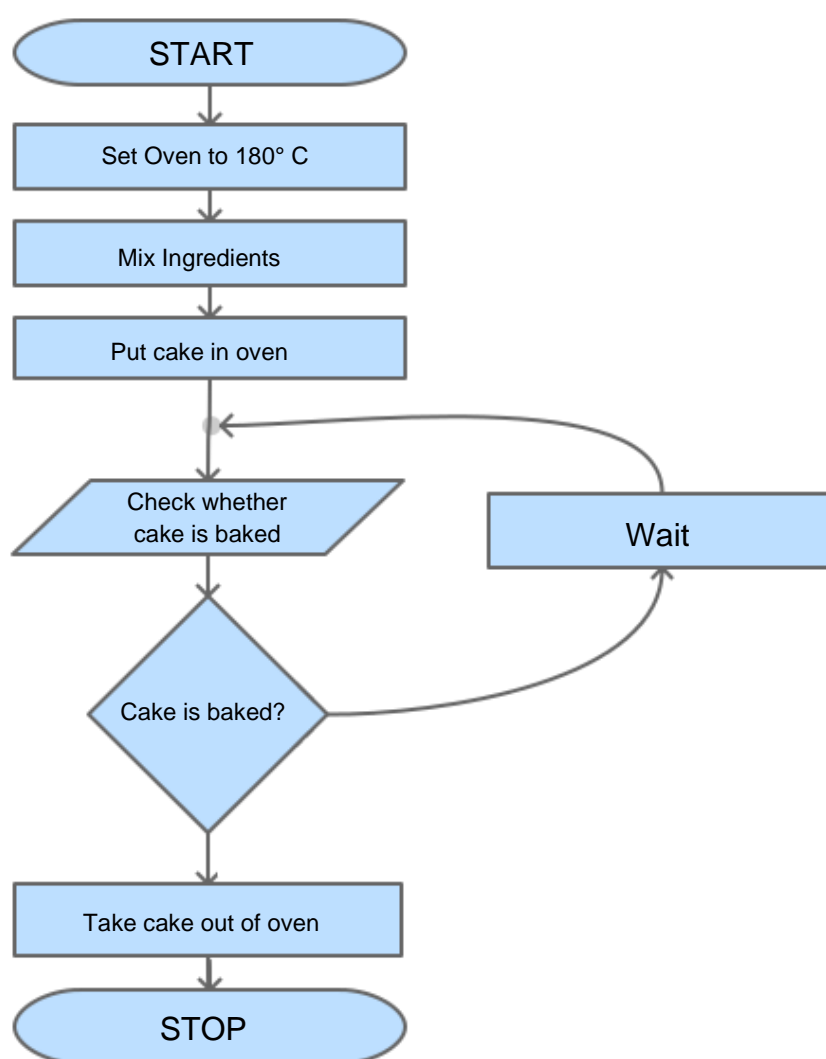
Flowchart for a numerical trick

This is a flowchart for a numerical magic trick. This flowchart shows a simple sequence, with no decision boxes. We begin the process from the Start box and continue following the flow lines completing the instructions in each of the boxes until we reach the Stop box.

Example: Flowchart for Baking a Cake

This flowchart represents an algorithm for baking a cake. It has a decision box in it. The decision box determines whether the cake is ready to take out of the oven, and if not instructs the chef to wait until it is.

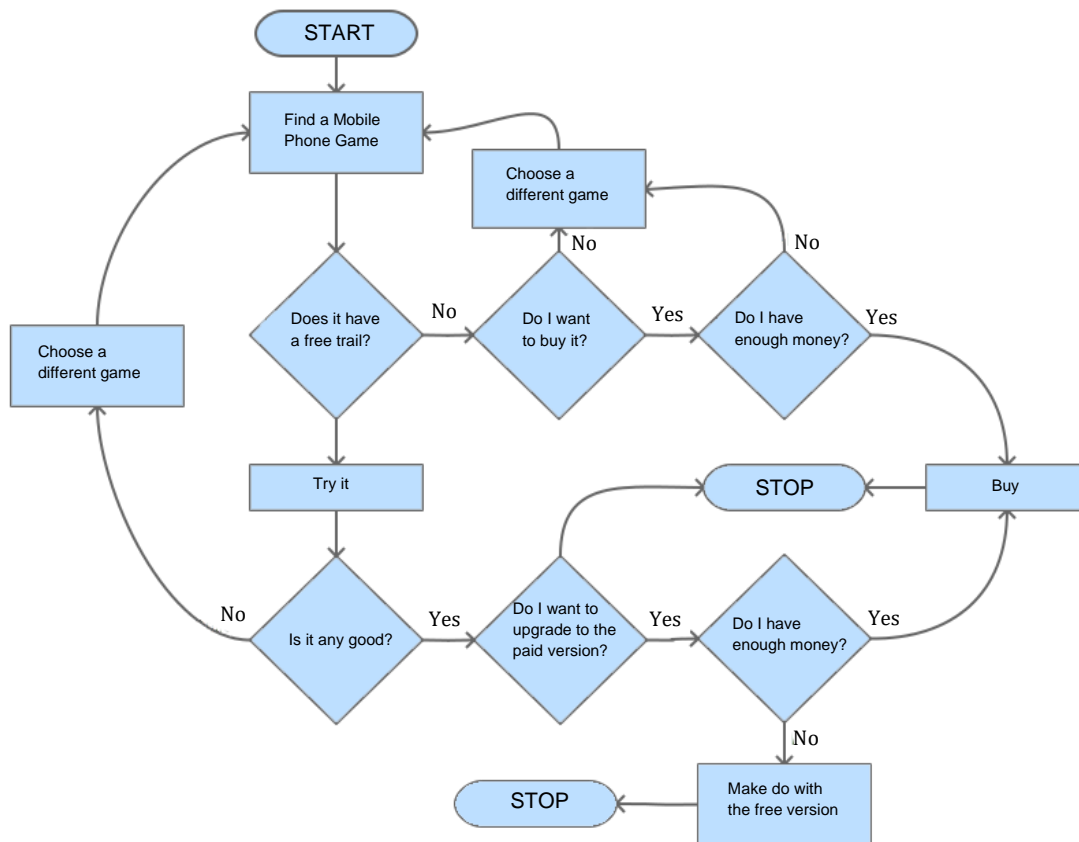
“Is cake baked?” is an input to the algorithm and has the input/output shaped box.



Flowchart for baking a cake

Example: Flowchart for choosing a Mobile Phone Game

This flowchart shows the sequence of actions for choosing a mobile phone game. Each of the diamond shaped boxes has a question in it. The answer to the question can be yes or no. The answer determines which box to go to next.



Flowchart for choosing a mobile phone game

Example: Flowchart for Getting up in the Morning

Make a flowchart for getting up in the morning, having breakfast, and getting to school.

HINTS:

1. Start with a sequence of actions and write those steps as pseudocode, in an ordered list.
2. Take some of the steps and decompose them into smaller steps detail. For example, if you have a step called 'make breakfast' - that can be broken down into smaller steps like "put bread in toaster".
3. Represent this more detailed sequence as a flowchart.

4. Make the flowchart more interesting, by introducing variation. For example, how does the weather affect getting to school? The variations may depend on questions like:
 - Is it raining?
 - Am I late for school?
5. Add decision boxes for these questions, with different steps if the answer is yes or no.

3.4 PSEUDOCODE



Concepts

Instead of creating a flowchart, an algorithm can be represented in pseudocode.

Pseudocode is a way to informally describe how an algorithm operates. It combines informal natural language (e.g. English) with some of the structure of programming languages.



Steps

Example: Pseudocode for the 347347 'Magic Trick'

The numerical magic trick flowchart could be written as pseudocode as follows:

```
Think of a 3 digit number like 347
Write the number out twice to make a 6 digit number like 347347
Divide by 13
Divide by 11
Divide by 7
You should be back to your original number.
STOP
```

Example: Pseudocode for Baking a Cake

Pseudocode for baking a cake is shown below:

```
Set oven to 180°C
Mix ingredients
Put cake in oven
Check whether cake is baked and if it isn't
    Wait
    Take cake out of oven
STOP
```

Pseudocode is written using the structure and some conventions of programming languages. The word 'Wait' is **indented**, which indicates that it only happens while the cake isn't baked.

3.5 FIXING ALGORITHMS



Concepts

When writing algorithms it is easy to make mistakes such as leaving out steps, putting steps in the wrong order, or making incorrect decisions. These errors must be corrected.

Here are some common errors and how to fix them.

1. Incorrect Sequence

Writing instructions in the wrong order is known as an incorrect sequence.

Here is an algorithm for brushing teeth, written in pseudocode. The instructions are in the wrong order. The algorithm can be fixed by moving 'Put toothpaste onto toothbrush' to the top.

```
Brush Top Right Teeth
Brush Bottom Right Teeth
Brush Top Left Teeth
Brush Bottom Left Teeth
Put toothpaste onto toothbrush
STOP
```

2. Incorrect Decision Outcome

Here is an algorithm for baking a cake and taking it out of the oven when it is baked and not before.

```
Set oven to 180°C
Mix ingredients
Put cake in oven
Is cake baked?
    Wait
Take cake out of oven
STOP
```

The decision to wait could be wrong because there is no test to see if the cake is actually baked. To correct a possible incorrect decision outcome, an extra line of pseudocode is needed:

```
Set oven to 180°C
Mix ingredients
Put cake in oven
Test with Fork
Is cake baked?
    Wait
Take cake out of oven
STOP
```

3. Missing Program Element

Here is an algorithm for baking a cake which is missing some important steps.

```
Set oven to 180°C.
Wait for oven to heat up.
Take cake out of the oven.
STOP
```

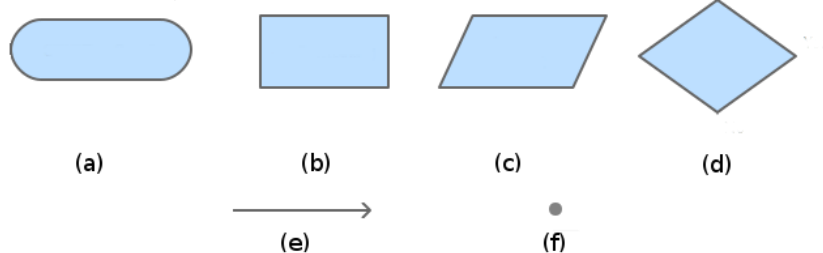
The algorithm can be fixed by adding the missing steps:

```
Set oven to 180°C.
Mix ingredients
Put the mixture into the cake tin
Wait for oven to heat up.
Put cake tin into oven
Wait for cake to cook.
Take cake out of the oven.
STOP
```

3.6 REVIEW EXERCISE

1. A sequence is:
 - a. A number of instructions that can be obeyed in any order.
 - b. A number of instructions that should be obeyed one after the other.
 - c. An analysis of a problem to be solved.
 - d. A collection of recommendations for enhancing a computer program.
2. Which of these is not used to represent an algorithm?
 - a. A computer program.
 - b. Flowchart.
 - c. Pseudocode.
 - d. Pseudoscience.

3. Match the following symbols to their names:



Arrow	
Decision	
Start or Stop	
Process	
Input or Output	
Connector	

4. In the following pseudocode which instruction is obeyed immediately after 'Mix Ingredients'?

```

Set oven to 180°C
Mix ingredients
Put cake in oven
Check whether cake is baked and while it isn't
    Wait
Take cake out of oven
STOP
  
```

- a. STOP
 - b. Wait
 - c. Take gingerbread out of oven
 - d. Put cake in oven
5. The following program is supposed to drain water from a washing machine. What error does it have?

```

Turn on drainage pump
  
```

```
Check water level
While there is no water left
    Wait
Turn off drainage pump.
STOP
```

- a. Missing instruction
- b. Incorrect sequence
- c. Incorrect decision outcome
- d. Incorrect indentation

LESSON 4 - GETTING STARTED

After completing this lesson, you should be able to:

- Start and run a program
- Enter code
- Create and save a program
- Open and run a program

4.1 INTRODUCING JAVASCRIPT



Concepts

JavaScript is a dynamic computer programming language. It is lightweight and most commonly used as a part of web pages, whose implementations allow client-side script to interact with the user and make dynamic pages. It is an interpreted programming language with object-oriented capabilities.

These learning materials uses a free online code compiler, which allows you to write, edit and run code.

JavaScript is called an interactive programming language, which means that when you enter a command the system evaluates the command, obeys it and prints or displays the result.

4.2 EXPLORING JAVASCRIPT



Concepts

Launch JavaScript

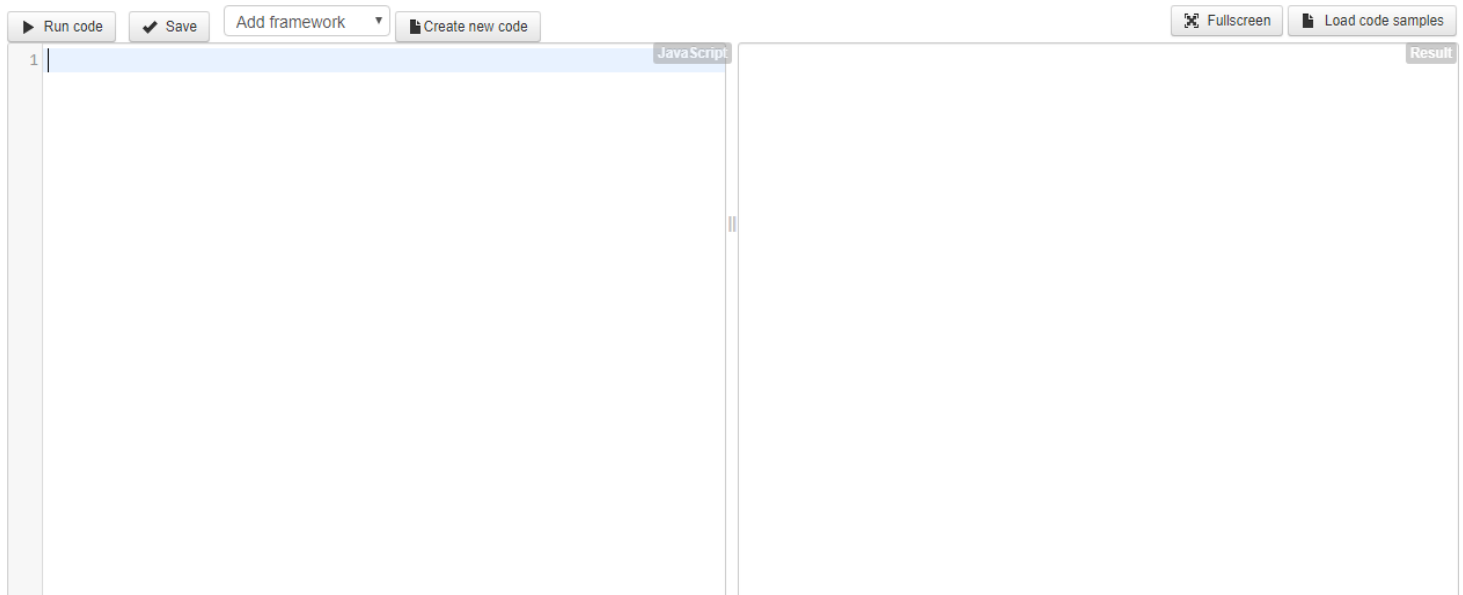
1. All that you need to get started writing JavaScript code is a simple text editor, such as Notepad or you can use a free online JavaScript editor
2. Go to <https://js.do/> It's an Online JavaScript Editor. Select the button Create new code.

JS.do Online JavaScript Editor

[login or register \(free!\)](#)

"Edit your code online. Simple, light and fast!"

Code address: <https://js.do/code/> Description:

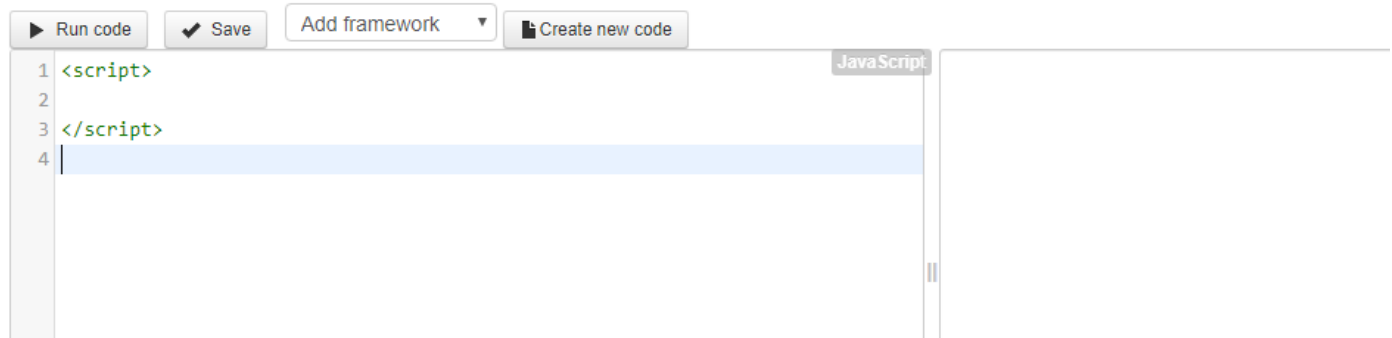


- The `<script>` tag alerts the browser program to start interpreting all the text between these tags as a script. The `// Edit your script here` is just a comment.

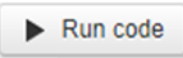
JS.do Online JavaScript Editor

"Edit your code online. Simple, light and fast!"

Code address: <https://js.do/code/> Description:



- Type `alert("First Steps in Coding");`

- Click the button Run Code  and view the result. Depending on what browser you are using, a popup box would look like.

First Steps in Coding

OK

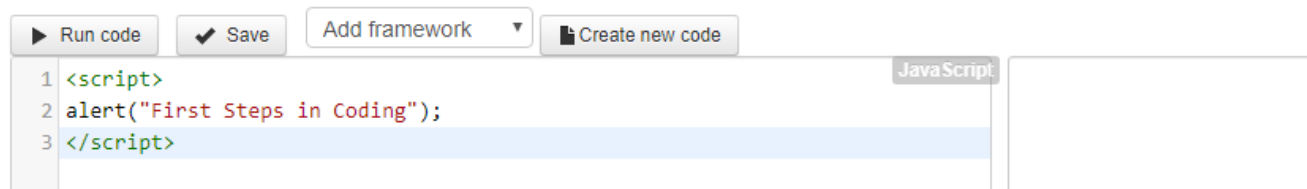
How the program works

- In the previous example, JavaScript understood and acted on the code

JS.do Online JavaScript Editor

"Edit your code online. Simple, light and fast!"

Code address: <https://js.do/code/> Description:



which instructs it to display the text inside the inverted commas onto the screen.

- The result is that the words **First Steps in Coding** are displayed on screen.

- In JavaScript, the **alert** command is used to output information to the screen through a popup message with 'Ok' button.

Let's try another instruction:

1. Go to <https://js.do/>.
2. Select the button Create new code
3. The tags `<script>` and `</script>` will appeared on left pane.
4. Type **asdf**

JS.do Online JavaScript Editor

"Edit your code online. Simple, light and fast!"

Code address: <https://js.do/code/> Description:

```
1 <script>
2 asdf
3 </script>
```

JavaScript

5. Click the button Run Code
6. View the result. In this example, JavaScript does not understand the letters **asdf**. and provides **error messages** when it doesn't understand what has been typed in

JS.do Online JavaScript Editor

"Edit your code online. Simple, light and fast!"

JavaScript error: Uncaught ReferenceError: asdf is not defined on line 2

Code address: <https://js.do/code/> Description:

```
1 <script>
2 asdf
3 </script>
```

JavaScript

4.3 SAVING A PROGRAM



Concepts

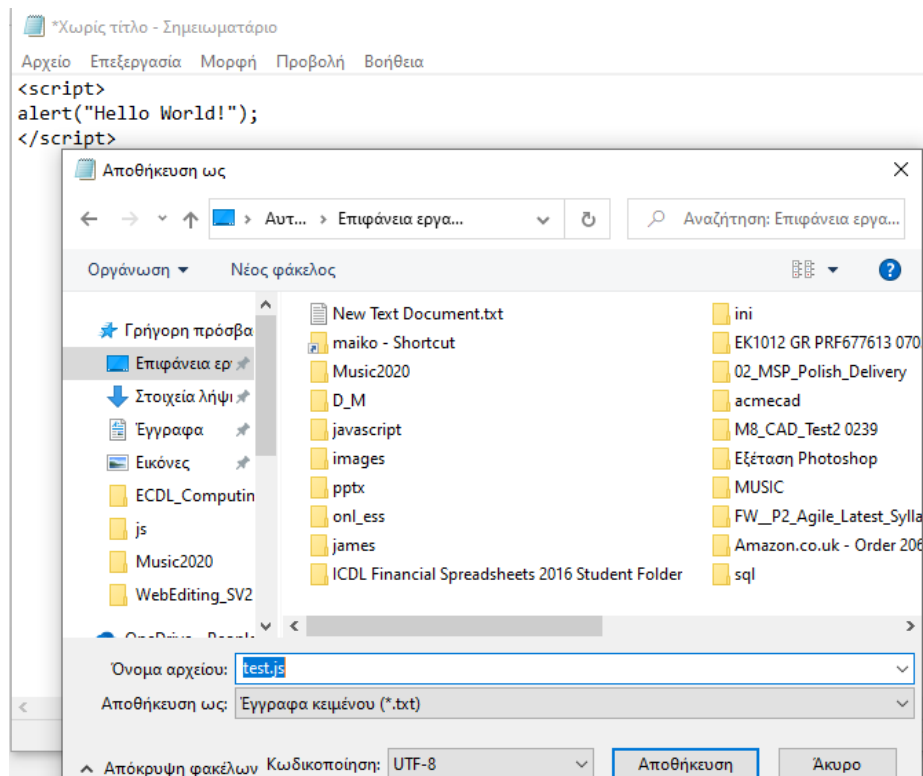
When you write some code, it is good practice to save it so that you can use it again. JavaScript programs have the file extension **.js**. The **.js** extension at the end of the name tells the computer it is a JavaScript program, for example, **MagicTrick.js**

Create and Save a Program

1. Go to <https://js.do/>
2. Select the button Create new code.
3. The tags `<script>` and `</script>` will appeared on left pane.
4. Type **alert("Hello World!");**
5. Start Notepad. Copy the code from the browser and paste it in the Notepad.

```
<script>
alert("Hello World!");
</script>
```

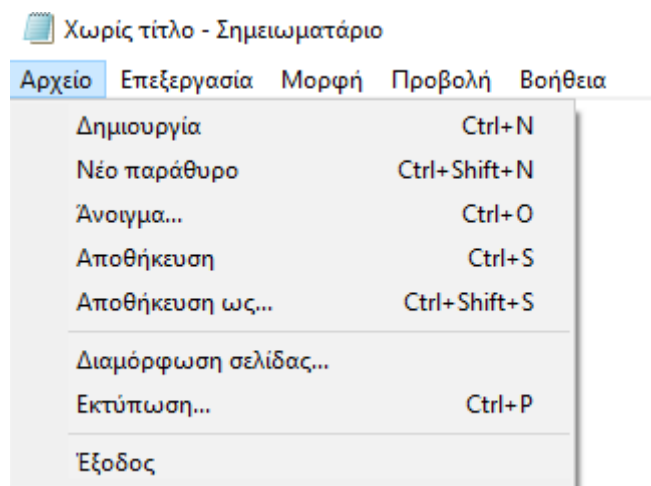
6. Click **File, Save As** and observe the dialogue box. Notice:
7. In the file name dialog that appears, type **test.js**.



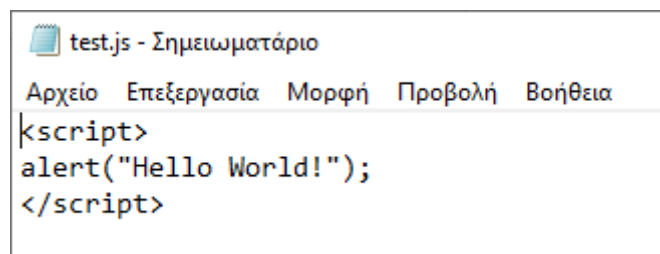
8. Click the **Save** button.
9. Click the X to close the window and the program.

Open and Run an Existing Program

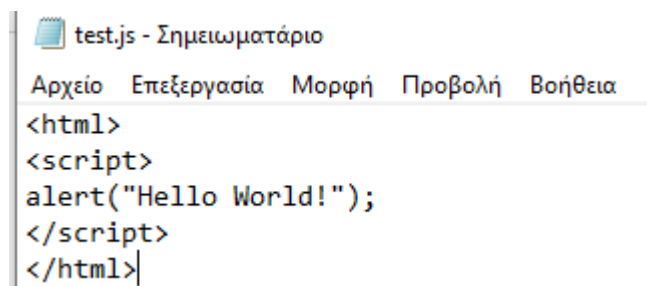
1. Start Notepad
2. Click **File, Open**.



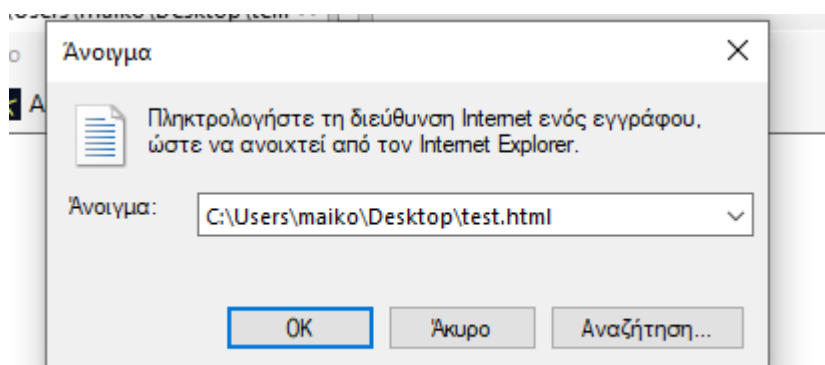
3. In the dialog that appears, select the file name **test.js**
4. Click the **Open** button. The saved program will open.



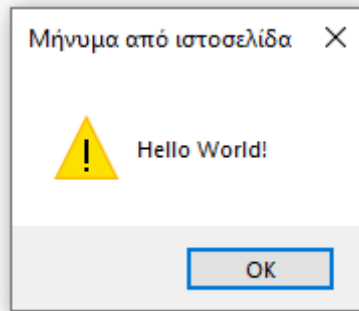
5. Type the tags `<html>` in the first line and `</html>` in the last line of the test.js file.



6. Click **File, Save As**, in the file name dialog that appears, type **test.html**
7. To run the program, open Internet Explorer.
8. Click **File, Open**, in the dialog that appears, select the file name **test.html**



9. The result is that the words Hello World! are displayed on screen.



10. Close the results window by clicking **OK**

4.4 REVIEW EXERCISE

1. A _____ file is a text file containing JavaScript code that is used to execute JavaScript instructions in web pages?
 - a. .js
 - b. .jvs
 - c. .jv
 - d. .html

LESSON 5 - PERFORMING CALCULATIONS

After completing this lesson, you should be able to:

- Recognise and use arithmetic operators. $+$, $-$, $*$ and $/$
- Know how parentheses affect the evaluation of mathematical expressions
- Understand and apply the precedence of operators in complex expressions
- Understand how to use parenthesis to structure complex expressions

5.1 PERFORMING CALCULATIONS WITH JAVASCRIPT



Concepts

Operators

In common with most computer languages, JavaScript can perform mathematical calculations, or evaluate mathematical expressions, for example:

$$10+12+15.$$

It uses the symbol `*` for multiply and `/` for divide rather than using `x` and `÷`.

NOTATION	MEANING
$3 * 4$	3 multiplied by 4
$3 / 4$	3 divided by 4
$3 + 4$	3 plus 4
$3 - 4$	3 minus 4

In JavaScript:

7 multiplied by 9 is written as `7 * 9`

63 divided by 3 is written as `63 / 3`

The spaces between the numbers are optional and `7*9` will work just as well as `7 * 9`.

The mathematical symbols like `*`, `/`, `+` and `-`, used in programming to perform calculations are called **operators**.

Parentheses Matter

Mathematical expressions in JavaScript can also contain parentheses, also known as brackets, for example:

$$10-(6-4)$$

If we evaluate this expression from left to right, ignoring the parentheses, the solution would be 0.

$$10-(6-4) = 0$$

However, this is incorrect.

Parentheses indicate what should be calculated first. In JavaScript, as in maths, what is inside the parentheses is calculated first. To evaluate an expression like $10-(6-4)$ JavaScript breaks it down as follows:

First: $6-4=2$

Then: $10-2=8$

So, $10-(6-4)=8$

5.2 PRECEDENCE OF OPERATORS



Concepts

When there are parentheses in an expression, it is clear to JavaScript what order to calculate expressions. When there aren't parentheses, there is an accepted order of operations. A rule of formal language determines the order in which the operators are applied. The rule is called the **Precedence of Operators**. In most computer languages the sequence or order in which operators are applied is multiplication, division, addition, subtraction.

Operators $*$ and $/$ are applied before operators $+$ and $-$.

Here are some examples:

$1+7+2+3$

$1+7 = 8$, $8+2 = 10$, $10+3 = 13$ answer: 13 Correct

$2*2*3+4$

$2*2 = 4$, $4*3 = 12$, $12+4 = 16$ answer: 16 Correct

$4+3*2=?$

You might think the answer should be 14, because:

$4+3 = 7$, $7*2 = 14$

However, because of precedence of operators, multiplication is done first. $4+3*2$ is calculated exactly as if it had been written $4+(3*2)$. So:

$4+(3*2) = 4+6$, and $4+6 = 10$ answer: 10 Correct

$31*7+112*2$

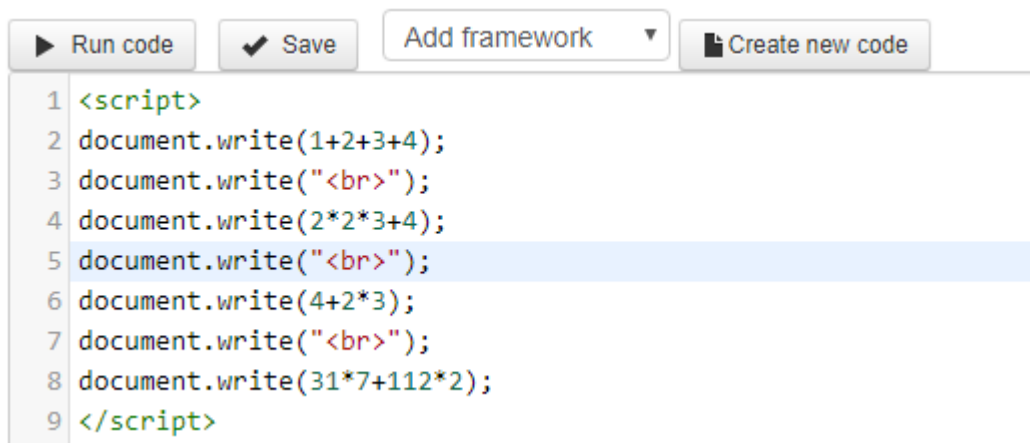
The two multiplication parts are done first, then the addition.

$31*7+112*2 = 217+224$, and $217+224 = 441$

Using JavaScript as a Calculator

Do the same calculations as in the 'Precedence of Operators' section using JavaScript:

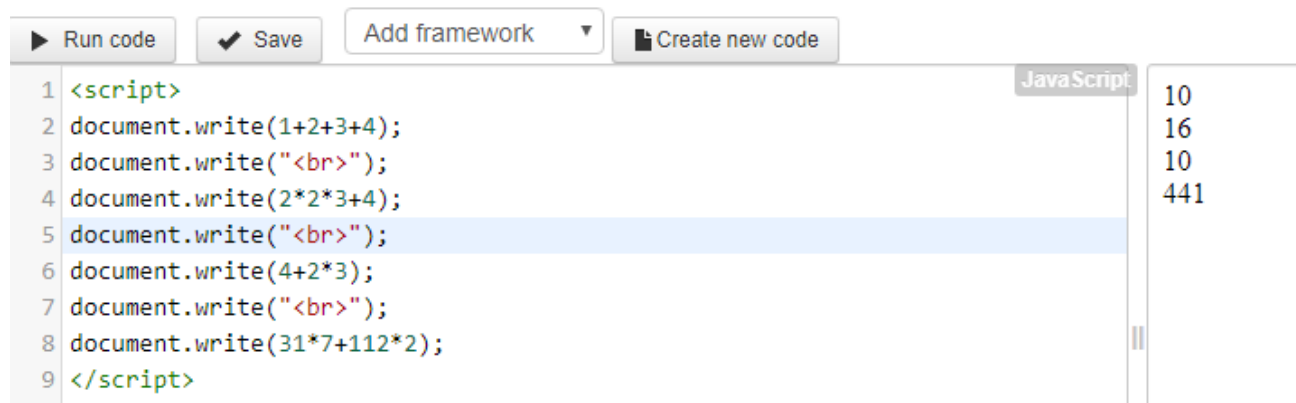
1. Go to <https://js.do/>
2. Select the button Create new code.
3. The tags `<script>` and `</script>` will appeared on left pane.
4. Type in the expressions (calculations)



The screenshot shows the js.do web editor interface. At the top, there are buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. Below these buttons is a text area containing the following JavaScript code:

```
1 <script>
2 document.write(1+2+3+4);
3 document.write("<br>");
4 document.write(2*2*3+4);
5 document.write("<br>");
6 document.write(4+2*3);
7 document.write("<br>");
8 document.write(31*7+112*2);
9 </script>
```

5. Click the button Run Code and check the results on the right pane



The screenshot shows the js.do web editor interface after the code has been executed. The 'Run code' button is highlighted. On the right side of the editor, the results of the calculations are displayed:

JavaScript	Results
1 <script>	10
2 document.write(1+2+3+4);	16
3 document.write(" ");	10
4 document.write(2*2*3+4);	441
5 document.write(" ");	
6 document.write(4+2*3);	
7 document.write(" ");	
8 document.write(31*7+112*2);	
9 </script>	

Example: Calculating the cost of tins of paint

If red paint costs €31 per litre and gold paint costs €112 per litre, then the cost of 7 litres of red paint and 2 litres of gold paint can be written as:

$$31*7+112*2$$

In this case, you can see that it is important to calculate what each colour paint costs first, and then getting the total cost.

So, $(31*7)+(112*2)$ gives the correct overall price, whereas evaluating from left to right does not.

Parentheses are not needed here because of the order of operations.

However it is always possible to put additional parentheses in to override the order of operations. Parentheses have priority over any rules about order of operations.

The words are printed to the browser because we used **document.write**. This is another method we will use (besides alert) that will help us quickly test something out.

The **br** tag is used to insert a line break into text , so using **document.write("
");** you can insert a single line break. The **
** tag is an empty tag which means that it has no end tag.

5.3 REVIEW EXERCISE

1. How do you get JavaScript to evaluate an expression?
 - a. Type 'evaluate', and then type in the expression.
 - b. Break it down into smaller pieces first.
 - c. Type in the expression after the >>> prompt.
 - d. Use a calculator instead.

2. Which of the following expressions is used to multiply two numbers in JavaScript?
 - a. $4 + 7$
 - b. 4×7
 - c. $4 \# 7$
 - d. $4 * 7$

3. Evaluate the expression $3*4*2+8$ using the Operator Precedence rule and select the appropriate answer below:
 - a. 24
 - b. 32
 - c. 104
 - d. 120

LESSON 6 – DATA TYPES AND VARIABLES

After completing this lesson, you will be able to:

- Use different data types, character, string, integer, float, Boolean
- Define the programming construct term variable. Outline the purpose of a variable in a program
- Define and initialise a variable
- Assign a value to a variable
- Use data input from a user in a program
- Use data output to a screen in a program

6.1 DATA TYPES



Concepts

In computing all kinds of data are used in order to solve problems and create information. The type of data determines how it is stored in the computer's memory and what can be done with it.

For example, your age is stored as a number and we can perform mathematical calculations on it, like compare it to that of your friend. And your name is stored as text.

In computing data types are used to determine how to store data and what operations to carry out on the data.

The JavaScript **data types** you will use most often are: **Numerical, String, and Boolean.**

Data Types in JavaScript

Numerical

The number data type is used to represent positive or negative numbers with or without decimal place, or numbers written using exponential notation e.g. 1.5e-4 (equivalent to 1.5x10⁻⁴).

String

Describes or defines a piece of text, or 'string' of characters. For example the name of a person, such as 'John Smith'. String values often arise from using the input() command.

Boolean

Describes or defines a value which is either True or False. Boolean data types can only ever be one of these two values. Boolean values most often arise from comparison of numbers.

For example the expression:

3 < 4 gives the boolean value True.

6.2 VARIABLES



Concepts

A **variable** is used in code to represent a piece of data so that the data can be used several times in a program. A variable is like a placeholder for actual values. It can hold the value and then retrieve it for use later.

Assigning a value to a variable

A variable can represent different types of data or values, such as a number or a string. In the code you need to specify what value the variable will represent and this is known as assigning a value to a variable.

Before you can use a variable, you should declare its existence to the JavaScript engine using the **var** keyword. This warns the engine that it needs to reserve some memory in which to store your data later.

In the following example, the value **3** is assigned to the variable named **x**.

```
var x=3
```

In this case, each time the variable **x** is used in the program it will represent **3**.

A variable can also hold a string, which is a collection of characters. In the following example, the value **Good Morning**, which is a string rather than a number, is assigned to the variable named **Greeting**.

```
var Greeting = "Good Morning"
```

In this case, each time the variable **Greeting** is used in the program it will represent **Good Morning**. After the assignment the variable name **Greeting** can be used in place of the string 'Good Morning'.

Using `document.write()` for Output

JavaScript outputs the contents of a variable to the screen, using the **document.write()** method.

So in the example `var x=3`

The command **document.write(x)** will print the value **3** to the screen.

The command **document.write(x+2)** will print the value **5** to the screen.

And in the example `var Greeting = "Good Morning"`

The command `document.write(Greeting)` will print the value **Good Morning** to the screen.

Defining, Initialising and Updating Variables

Defining Variables

Defining a variable means defining in the code what data type the variable will hold, for example, a number (e.g. integer, float) or a string. In JavaScript you do this the first time you assign a value to a variable. JavaScript determines that the variable will hold the data type of the assigned value.

If you assign the number 3 (i.e. the value) to the variable x, JavaScript will assume that x will always represent a numeric value.

`var x=3`

If you assign the string 'country' to the variable q, you are telling JavaScript that q will always represent a string. You use single or double quotes around the text to indicate that it is a string data type.

`var q="country"`

In the example, the piece of text is enclosed in double quotes, which indicates that it is the string data type. Any characters enclosed in inverted commas are considered a string, even a number. In the following example 12 is a string not a number:

`var Age = "12"`

Initialising Variables

You need to assign an initial value to a variable before you can use it. The first time a value is assigned to a variable is called **initialisation of the variable**.

If you try to apply a command to a variable before it has been initialised you will get an error message. Before a value is assigned to a variable, the variable is known as **uninitialised**.

Updating Variables

The value of a variable can be updated by assigning a new value to it. You update the variable and it is automatically updated wherever it is used throughout the program. This is an important reason for using variables as you only have to make one update to update multiple occurrences. A variable always holds the most recent value that has been assigned to it.

Consider a variable called `x` that is currently 3. The assigned value can be changed to 7 by typing the code:

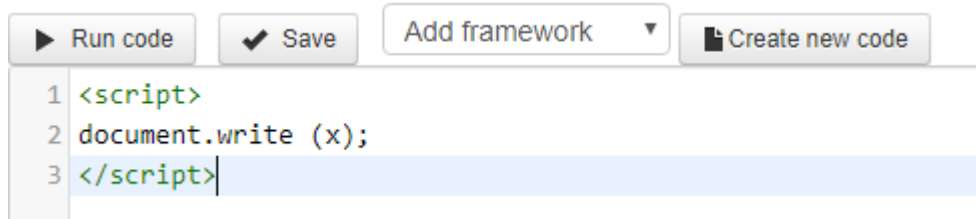
`var x=7`

The command **`document.write(x)`** will now print the value **7**.

The command **`document.write(x+2)`** will now print the value **9**.

Example: Working with Variables

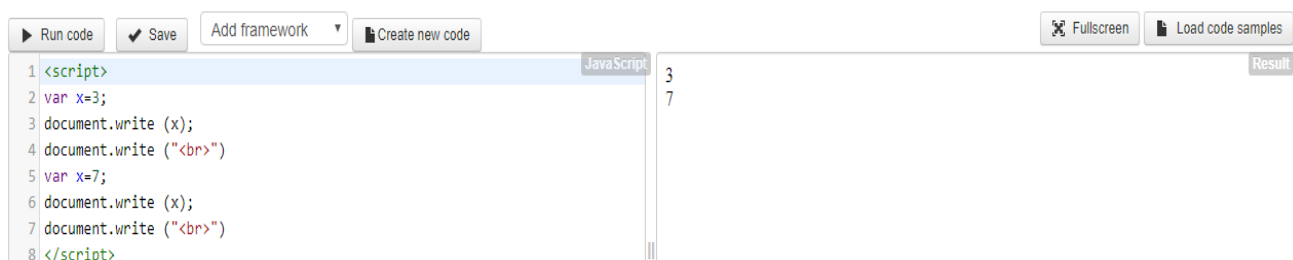
1. Go to <https://js.do/>
2. Select all the code lines on the left pane and press delete.
3. Click the button Run Code and the tags `<script>` and `</script>` will appeared on left pane.
4. Before initialising the variable `x`, type in the command `document.write (x)`



5. Observe the error message.

JavaScript error: Uncaught ReferenceError: x is not defined on line 2

6. Initialise the variable `x` to 3.
`var x=3;`
7. Type the `document.write` command again.
`document.write (x);`
8. Update the variable `x` to 7.
`var x=7;`
9. Print the new value.
`document.write (x);`



6.3 BEYOND NUMBERS



Concepts

Working with Strings

As well as adding numbers together, you can join strings and characters together. To do this in JavaScript, you can use the + symbol.

For example, initialise the variable **Name** to “**David**”, and **Question** to “**What is your age?**”

```
Name = "David"
```

```
Question = "What is your age "+Name+"?"
```

This returns the result:

```
What is your age David?
```

Note the blank spaces in the string. Without the spaces at the ends of the string there would be no space between the word you and the word David and the question would look like this:

```
What is your ageDavid?
```

Changing strings to numbers

Sometimes a string needs to be converted to a number (e.g. an integer or float). For example if the variable called Age is the string “12”,

```
Age = "12"
```

then Age +1 will give an error, because a number can't be added to a string.

To fix that, you can convert the variable Age to an integer as follows:

```
Age = Number( Age )
```

Using prompt () for User Input

You can write code that asks a user to enter some information in the form of a string, for example their name.

In JavaScript you use the **prompt ()** command to prompt the user to enter a string

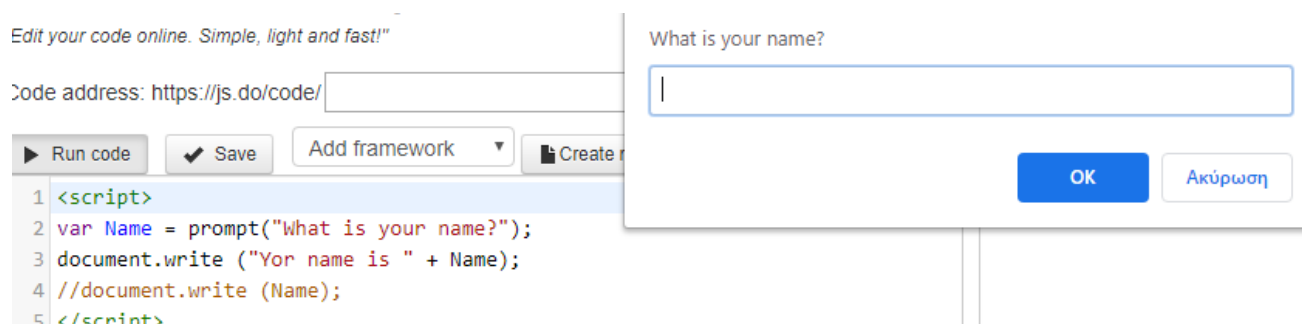
The **prompt () command** is an instruction in JavaScript that requests the user to enter information and the program waits until the information is entered.

Example: Input and Output a Name.

1. Go to <https://js.do/>
2. Select the button Create new code.
3. The tags <script> and </script> will appeared on left pane.
4. Type in the following code.

```
var Name = prompt("What is your name?");  
document.write ("Your name is " + Name);
```

5. Run the program.
6. Reply to the prompt “What is your name?” by typing in your name.
7. Press Enter.

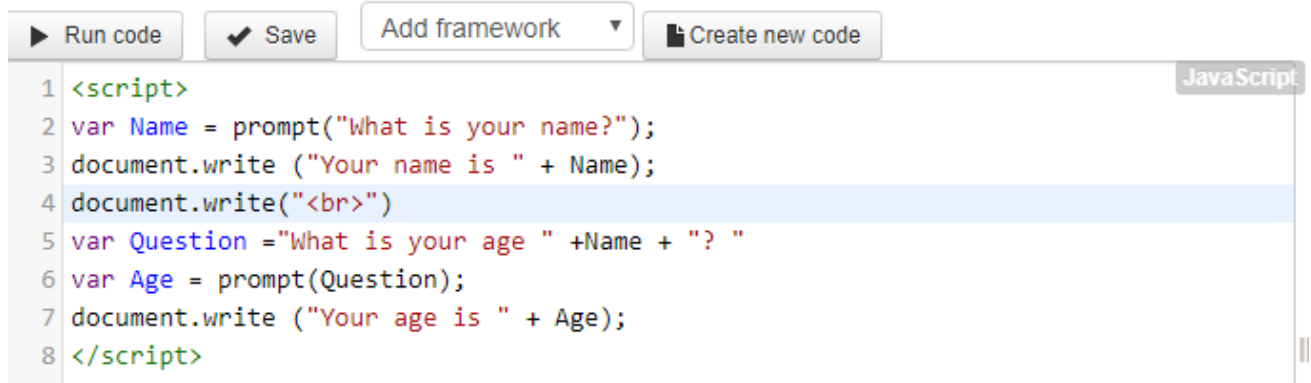


How the program works

- The user is prompted to enter a string, in this case their name, using the **prompt()** command.
- The program waits until the information is entered.
- The variable 'Name' is initialised to contain the name entered by the user.
- The **document.write()** command is used to output the text “Your name is:” followed by the contents of the variable 'Name' to the screen.

Example: Working with Strings

1. Modify the example so that it looks like this:



The screenshot shows a code editor interface with a toolbar at the top containing buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code is written in JavaScript and is as follows:

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 </script>
```

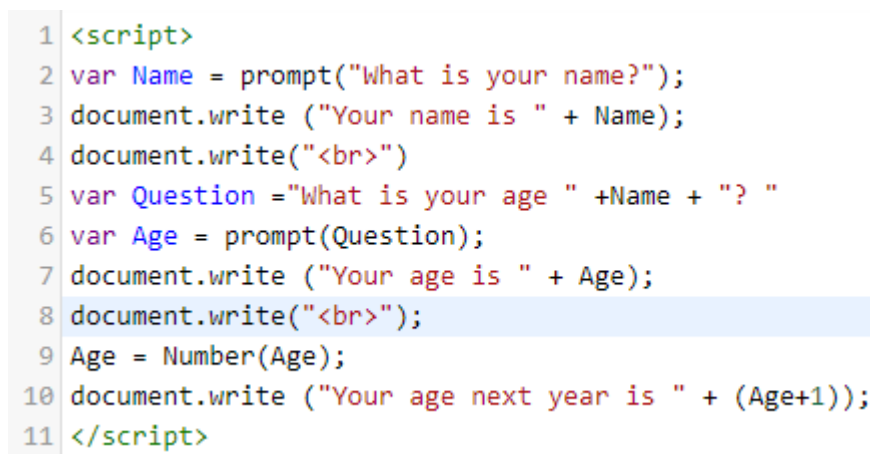
2. Run the program.
3. Answer the first question and press Enter.
4. Answer the second question and press Enter.



This screenshot shows the same code editor as before, but now the output of the program is visible on the right side. The code is identical to the previous one. The output displayed is:

```
Your name is Kate
Your age is 28
```

5. Modify the program so that it now reads:



The screenshot shows the code editor with the following modified JavaScript code:

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 document.write("<br>");
9 Age = Number(Age);
10 document.write ("Your age next year is " + (Age+1));
11 </script>
```

6. Now run the program again.
7. Answer the two questions.

▶ Run code

✓ Save

Add framework ▼

📄 Create new code

JavaScript

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 document.write("<br>");
9 Age = Number(Age);
10 document.write ("Your age next year is " + (Age+1));
11 </script>
```

Your name is Nick
Your age is 20
Your age next year is 21

6.4 REVIEW EXERCISE

1. Which statement about variables is correct?
 - a. Variables never change their value.
 - b. A variable name must use lower case letters only.
 - c. Variables must be initialised to zero.
 - d. Sometimes a variable can hold a Boolean value.
2. What is the purpose of a variable?
 - a. To make it easier to change a program in the future.
 - b. To tell JavaScript how to combine two numbers
 - c. To remember some value that will be used again later in the program.
 - d. To ensure that the source code is indented correctly
3. Which of these JavaScript statements initialises a variable?
 - a. Initialise().
 - b. var x=3.
 - c. start().
 - d. finish().
4. In JavaScript, if a variable named “length” has the value 200, and the instruction “length=400” is obeyed, what happens?
 - a. JavaScript gives an error message, because you can’t change the value of a variable.
 - b. JavaScript gives an error message, because you can’t change 200 to 400.
 - c. JavaScript assigns 400 to length and length now has the value 400.
 - d. JavaScript adds 400 into length and length now has the value 600.
5. In JavaScript, the prompt() function:
 - a. Is used when you want to input a USB key
 - b. Can be used with a prompt to make input faster, or without a prompt to make input slower.
 - c. Returns a string.
 - d. Must be the first function you call in a program.
6. In JavaScript, the document.write () function
 - a. Is used when you want to change the data type
 - b. Is used to display a result
 - c. Is the last command in a program.
 - d. Must be the first function you call in a program.
7. A data type which has only two values is called
 - a. Integer

- b. Float
- c. String
- d. Boolean

LESSON 7 – TRUE OR FALSE

After completing this lesson, you should be able to:

- Use Boolean logic expressions in a program
- Recognise types of Boolean logic expressions to generate a true or false value like: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT
- Understand the precedence of operators and the order of evaluation in complex expressions

7.1 BOOLEAN EXPRESSIONS



Concepts

In everyday life, you often check if something is **True** or **False** before deciding what to do next. This concept is also used in computer programs where conditions are checked to see if they are **True** or **False** before decisions are made on what to do next. For example:

Is it cold outside?

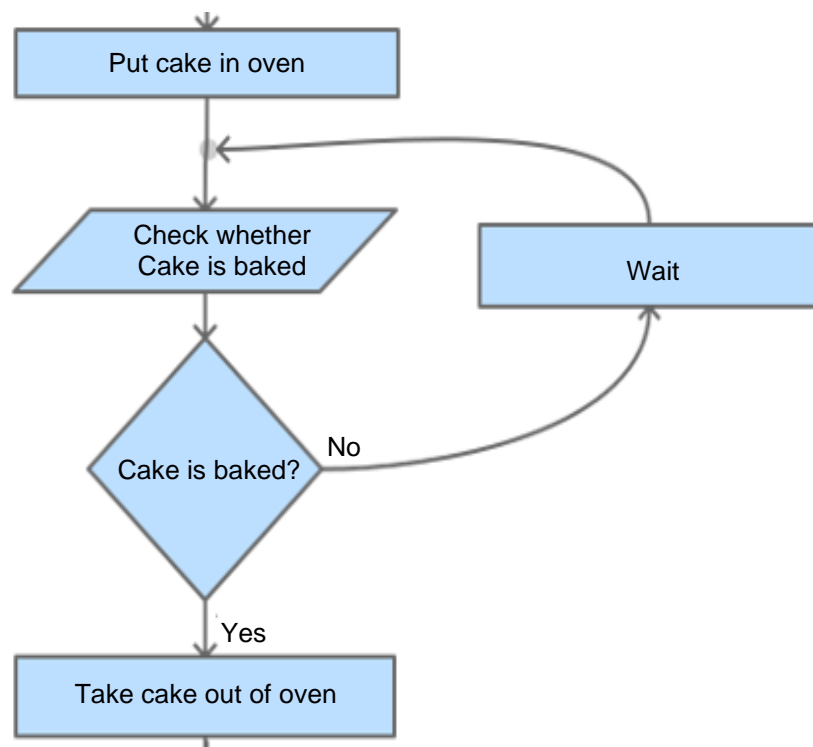
True – wear a coat

False – leave the coat at home

Is the cake baked?

True – take it out of the oven

False – bake it for 5 more minutes and re-check



True or False in Flowcharts

In a flowchart the checks or tests are represented by decision boxes with outcomes **Yes** or **No**.

Computer programs work with the outcomes **True** and **False** rather than Yes or No.

True and False are important concepts in computing because we need a way to test if certain conditions exist in order to control what happens next in the program.

In computing, this logic test is called a **Boolean expression**. A Boolean expression results in a **Boolean value** that is either true or false.

One way in which JavaScript programs use Boolean expressions is to test the size of a numerical value against another value and give a 'True' or 'False' result. It carries out this test by using a **comparison operator**. A comparison operator compares the values on either side of them and decide the relation among them. They are sometimes referred to as relational operators.

$99 > 7$

Here the operator ' $>$ ' compares the values 99 and 7 to see if the value on the left is greater than the value on the right. If so, the result is True, otherwise the result is False.

So: $99 > 7$ is True but $6 > 7$ is False.

7.2 COMPARISON OPERATORS



Concepts

Comparison operators are a type of Boolean logic expression used to compare values and decide if the result is true or false.

Table of Comparison Operators

There are six comparison operators. In the table below X and Y are variables holding numerical values:

NOTATION	MEANING
$X > Y$	X is greater than Y
$X < Y$	X is less than Y
$X \geq Y$	X is greater than or equal to Y
$X \leq Y$	X is less than or equal to Y

<code>X == Y</code>	X is equal to Y
<code>X != Y</code>	X is not equal to Y

Note: Some other languages may use `<>` as not equal to and `=` as equal to. JavaScript 3.x doesn't use `<>` and it uses `==` instead of a single `=`.

Example: Comparison Operators

1. Go to <https://js.do/>
2. Select the button Create new code.
3. The tags `<script>` and `</script>` will appeared on left pane. Enter the examples below to check your understanding of the comparison operators.

```
1 <script>
2 document.write (15>4);
3 document.write("<br>");
4 document.write (5>6);
5 document.write("<br>");
6 document.write (22>=16);
7 document.write("<br>");
8 document.write (4<=16);
9 document.write("<br>");
10 document.write (4==16);
11 document.write("<br>");
12 document.write (14!=16);
13 document.write("<br>");
14 document.write (14==16);
15 document.write("<br>");
16 </script>
```

JavaScript

true
false
true
false
true
false
true
false

4. Make up your own examples. Anticipate what the answer to each should be. See whether the answers, True or False, are what you expect them to be.

7.3 BOOLEAN OPERATORS



Concepts

Boolean operators are the words **and (&&)**, **or (||)**, **not (!)** which are another Boolean logic expression used to combine Boolean values like True and False together to give a final result.

There are three basic Boolean operators:

&&

- Combines two Boolean values and gives a result - True if both values are True, otherwise False

<i>Expression</i>	<i>Result</i>
True and True	True
True and False	False
False and True	False
False and False	False

||

Combines two Boolean values and gives a result - True if either one or both values are True, otherwise False.

<i>Expression</i>	<i>Result</i>
True or True	True
True or False	True
False or True	True
False or False	False

!

Converts a single Boolean value from True to False, or False to True.

<i>Expression</i>	<i>Result</i>
not True	False
not False	True

Boolean expressions can include the use of both comparison operators and Boolean operators. This is an example:

`x < 3 && y < 12`

In such mixed expressions, the comparison operators are evaluated before the Boolean operators. That's because of the rules of operator precedence which we'll come back to later in this lesson.

Example: Boolean Expressions

1. Go to <https://js.do/>
2. Select the button Create new code.
3. The tags `<script>` and `</script>` will appeared on left pane.
4. Try to evaluate Boolean expressions in JavaScript.
5. Think of some combinations to make sure that you understand what `&&`, `||` and `!` do.

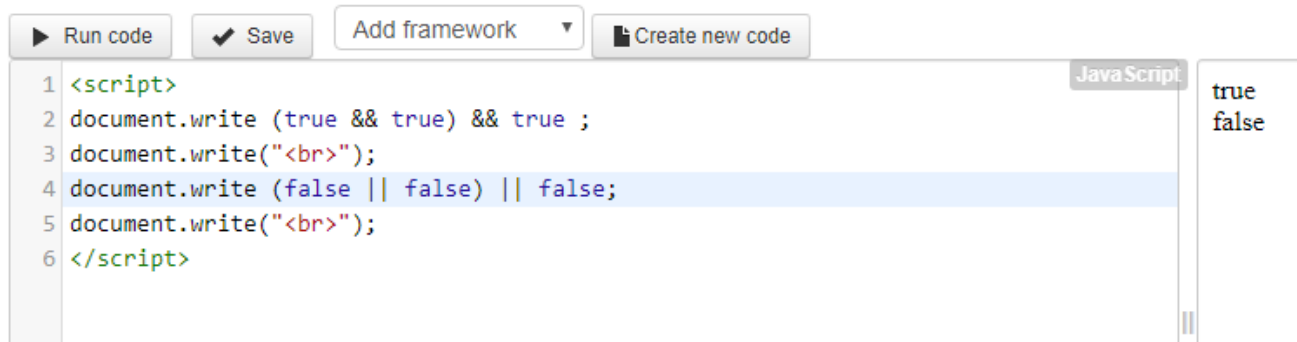


```
1 <script>
2 document.write (! true);
3 document.write("<br>");
4 document.write (! false);
5 document.write("<br>");
6 document.write (true && false);
7 document.write("<br>");
8 document.write (true || false);
9 document.write("<br>");
10 document.write (true && true);
11 document.write("<br>");
12 document.write (false || false);
13 document.write("<br>");
14 </script>
```

JavaScript

false
true
false
true
true
false

6. Build more complex Boolean expressions with parentheses, remembering that the expression within the parentheses is worked out first.

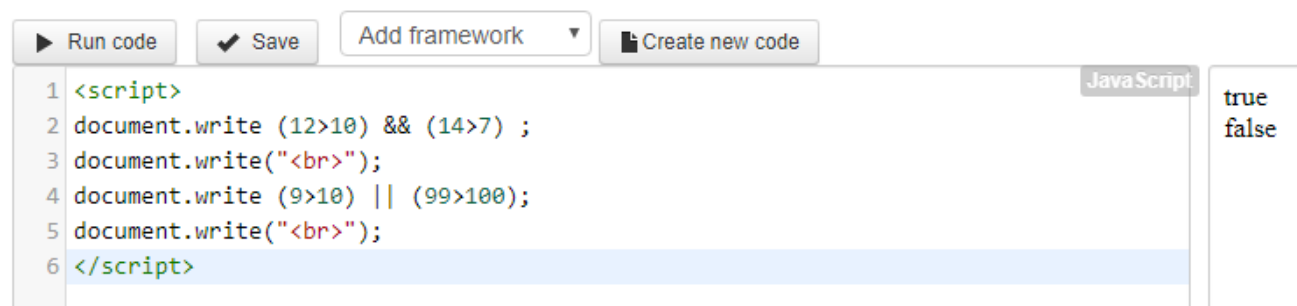


The screenshot shows a code editor with a toolbar at the top containing 'Run code', 'Save', 'Add framework', and 'Create new code'. The code area contains the following JavaScript code:

```
1 <script>
2 document.write (true && true) && true ;
3 document.write("<br>");
4 document.write (false || false) || false;
5 document.write("<br>");
6 </script>
```

On the right side of the editor, there is a 'JavaScript' tab and a console output area showing the results: 'true' and 'false' on separate lines.

7. Use JavaScript to evaluate some Boolean expressions that arise from comparing numbers.



The screenshot shows a code editor with a toolbar at the top containing 'Run code', 'Save', 'Add framework', and 'Create new code'. The code area contains the following JavaScript code:

```
1 <script>
2 document.write (12>10) && (14>7) ;
3 document.write("<br>");
4 document.write (9>10) || (99>100);
5 document.write("<br>");
6 </script>
```

On the right side of the editor, there is a 'JavaScript' tab and a console output area showing the results: 'true' and 'false' on separate lines.

7.4 BOOLEANS AND VARIABLES



Concepts

Initialising and assigning values to variables applies to numbers, strings and booleans.

Consider the instruction:

var Y = 200

This initialises the variable called Y to the numeric value 200.

Consider the instruction:

A = false

This initialises the variable called A to the Boolean value False.

Consider the instruction:

var name=prompt("What is your name?")

This initialises the variable called name to a string in response to the question, "What is your name?"

Here is what happens:

- i. The user is prompted with the words "What is your name?"
- ii. The user types in a string.
- iii. The string value is assigned to the variable called name.

The next example uses a Boolean variable. The variable is called **result**

Consider the instruction:

result = Age > 12

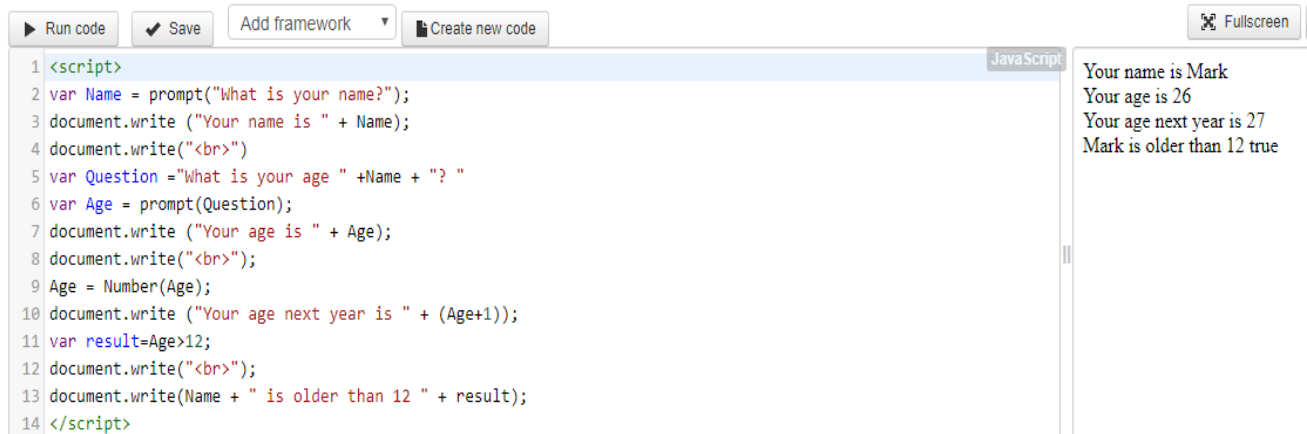
It compares Age with 12 and puts the Boolean result into the variable result.

Example: Working with Booleans

1. Open the file boolean.js, and paste it in browser as shown below.

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 document.write("<br>");
9 Age = Number(Age);
10 document.write ("Your age next year is " + (Age+1));
11 var result=Age>12;
12 document.write("<br>");
13 document.write(Name + " is older than 12 " + result);
14 </script>
```

2. Run the program.
Type in a name.
Type in some age greater than 12.



The screenshot shows a code editor with a toolbar at the top containing buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code is written in JavaScript and is as follows:

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 document.write("<br>");
9 Age = Number(Age);
10 document.write ("Your age next year is " + (Age+1));
11 var result=Age>12;
12 document.write("<br>");
13 document.write(Name + " is older than 12 " + result);
14 </script>
```

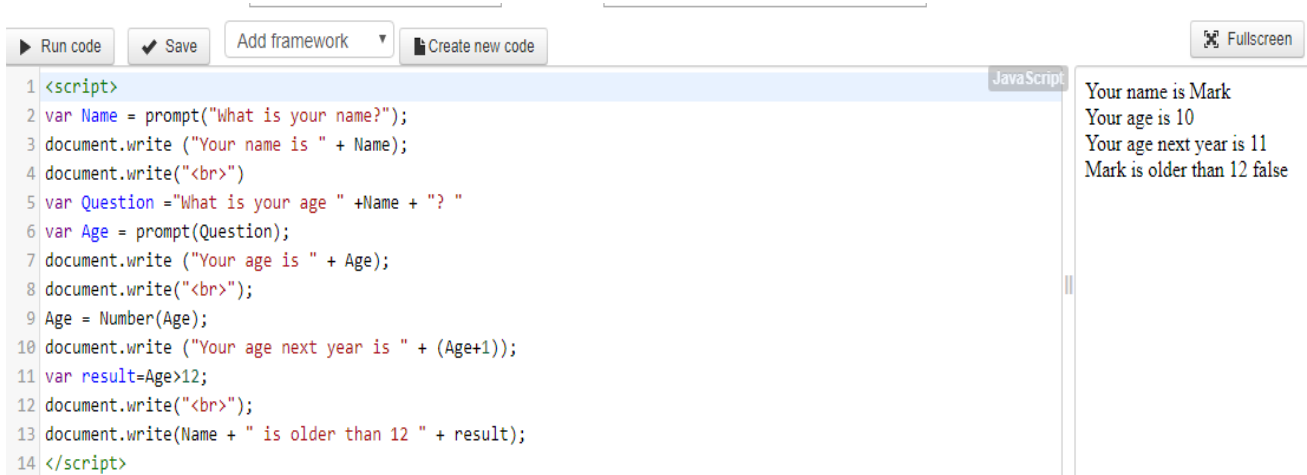
The output on the right side of the editor is:

```
Your name is Mark
Your age is 26
Your age next year is 27
Mark is older than 12 true
```

3. Run the program again.

Type in a name.

Type in some age less than 12 to see a different result.



The screenshot shows the same code editor as before, but with the output updated to reflect the new input. The code is identical to the previous screenshot:

```
1 <script>
2 var Name = prompt("What is your name?");
3 document.write ("Your name is " + Name);
4 document.write("<br>")
5 var Question ="What is your age " +Name + "? "
6 var Age = prompt(Question);
7 document.write ("Your age is " + Age);
8 document.write("<br>");
9 Age = Number(Age);
10 document.write ("Your age next year is " + (Age+1));
11 var result=Age>12;
12 document.write("<br>");
13 document.write(Name + " is older than 12 " + result);
14 </script>
```

The output on the right side of the editor is now:

```
Your name is Mark
Your age is 10
Your age next year is 11
Mark is older than 12 false
```

7.5 PUTTING IT ALL TOGETHER



Concepts

Parentheses in Boolean Expressions

We have seen in earlier lessons that parentheses are important in numerical expressions. They are also important in Boolean expressions.

(A && B) || C is not the same thing as **A && (B || C)**

Let's assume for example that A is False, B is True and C is True. Putting these values into the two expressions gives different results:

(A && B) || C evaluates to True

A && (B || C) evaluates to False

To confirm this, type in both Boolean expressions into JavaScript, using it as a calculator as shown in the next example.

Example: Parentheses in Boolean Expressions

1. Go to <https://js.do/>. Type in the following expressions:

```
1 <script>
2 var A=false;
3 var B=true;
4 var C=true;
5 document.write (A&&B);
6 document.write("<br>");
7 document.write ((A&&B)||C);
8 document.write("<br>");
9 document.write (B||C);
10 document.write("<br>");
11 document.write (A && (B||C));
12 document.write("<br>");
13 </script>
```

The output on the right shows the results of the expressions: false, true, true, false.

Note how the use of parentheses changes the result of the expression A and B or C.

Operator Precedence, including Booleans

The table below is a partial list of the order in which the most important operators are applied, or take precedence. In the list below, the operators at the top of the list are applied before operators at the end.

<i>Symbol</i>	<i>Operation</i>
* /	Multiply and divide
+ -	Addition and subtraction
<, >, <=, >=, ==, !=,	Comparison operators
!	Boolean not
&&	Boolean and
//	Boolean or

For example, in the absence of parentheses * is evaluated before +, as * is before + in the list. Likewise, '&&' is evaluated before '||', as '&&' is before '||' in the list.

An expression like:

$$A \parallel B \&\& C$$

Will be evaluated as:

$$A \parallel (B \&\& C)$$

A complex expression such as:

$$x < 2 \parallel 1+3*x > 2+4/2 \&\& ! x < y$$

Would be calculated as:

$$(x < 2) \parallel (1+(3*x) > 2+(4/2) \&\& (! (x < y)))$$

Even experienced programmers sometimes forget exactly what order the operators are evaluated in. In an expression as complex as the above it is usual to include some parentheses to make the intention clearer:

$$x < 2 \parallel ((1+3*x > 2+4/2) \&\& ! x < y)$$

The operator precedences for *, /, + and - are used so frequently that it is rare to use parentheses to clarify the order of evaluation.

7.6 REVIEW EXERCISE

1. Which of the following is a JavaScript expression to add three numbers together and divide the result by 3?
 - a. $4+5+6/3$
 - b. $4+5+6\div 3$
 - c. $(4+5+6)/3$
 - d. $(4+5+6)\div 3$
2. The expression ' $3 < x \ \&\& \ x \leq 7$ ' is true if x is:
 - a. 3,4,5,6 or 7
 - b. 2,3,4,5 or 6
 - c. 4,5, or 6
 - d. 4,5,6 or 7
3. In JavaScript, which expression could give a different result to the other three?
 - a. $a < b < c$
 - b. $c > b > a$
 - c. $a < b \ || \ b < c$
 - d. $a < b \ \&\& \ b < c$

LESSON 8 – ARRAY DATA TYPES

After completing this lesson, you should be able to:

- Understand arrays
- Use arrays in a program like

8.1 ARRAYS IN JAVASCRIPT



Concepts

As well as standard data types such as numbers, string and boolean, there is another category of data type called arrays. Arrays data types hold multiple items, for example, a list of names. Unlike most languages where array is a reference to the multiple variable, in JavaScript array is a single variable that stores multiple elements

The items in an array are called **elements**. Elements are referenced by numbers starting from zero.

If you have a list of items (a list of names, for example), storing the names in single variables could look like this:

```
var name1 = "Anna";
```

```
var name2 = "George";
```

```
var name3 = "Dennis";
```

The problem occurs if you want to loop through 1000 names and find a specific one. The solution is an array!

8.2 CREATING ARRAYS IN JAVASCRIPT



Concepts

The example that follows is our names variable that is initialized to an empty array: `var names = []`; The variable name is on the left, and a pair of brackets on the right that initializes this variable as an empty array.

To create non-empty arrays, place the items you want inside the brackets and separate them by commas:

```
var names = ["Mark", "Mary", "Nina", "Jimmy", "Clint"];
```

Inside an array, each item is assigned a number starting with zero. In the above example, Mark would have an index value of 0, Mary would have an index value of 1, Nina the index value of 2, and so on.

You can access an item from the array, by referring to the **index number**.

This statement accesses the value of the first element in name:

```
var name = cars[0];
```

Example:

```

1 <script>
2 var names = ["Mark", "Mary", "Nina", "Jimmy", "Clint"];
3 var name1 = names[0];
4 document.write(name1);
5 </script>

```

The screenshot shows a code editor with a toolbar at the top containing buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code is written in a light blue editor area. To the right of the code, there is a tab labeled 'JavaScript' and a preview area showing the output 'Mark'.

Working with ARRAYS in JavaScript

JavaScript has built in methods and properties for Arrays

Methods

push()	Adding elements at the end of an array.
unshift()	Adding elements at the front of an array
pop()	Removing elements from the end of an array
shift()	Removing elements at the beginning of an array

sort()	Sorts an array alphabetically or numerically.
reverse()	Reverses the order of the elements in an array.
concat()	Merges two or more arrays, and returns a new array.
slice(start,end)	Returns a new array made up of part of the original array. The first argument specifies where to start the selection and the second where to end the selection.
splice(start, itemcount, item1,,,itemX)	Adds/removes items to/from an array, and returns the removed item(s). The first argument specifies at what position to add or remove items, the second the number of items in the array to remove and the third the new items to the array. Arguments 2 and 3 are optional.

Example 1:

1. Go to <https://js.do/>
2. Type the code:

▶ Run code
✓ Save
Add framework ▼
Create new code

```

1 <script>
2 var names = ["Mark","Mary","Nina","Jimmy","Clint"]
3 document.write (names.length + "<br>"); // returns the length of the array
4 document.write (names[4] + "<br>"); // returns the 4th element of the array
5 names.push("James"); //adds new ellement at the end of the array
6 document.write (names.length + "<br>"); // returns the NEW length of the array
7 document.write (names[5] + "<br>"); // returns the 5th element of the array
8 names.unshift("Stacey"); //adds new ellement at the front of the array
9 document.write (names[0] + "<br>"); // returns the 1st element of the array
10 names.sort(); //sorts the array alphabetically
11 document.write (names[0] + "<br>"); // returns the 1st element of the array
12 document.write (names[6] + "<br>"); // returns the last element of the array
13 names.reverse(); // reverses the order of the elements
14 document.write (names[0] + "<br>"); // returns the 1st element of the array
15 document.write (names[6] + "<br>"); // returns the last element of the array
16 names.pop(); // removes element from the end of the array
17 document.write (names[6] + "<br>"); // the 6th element of the array does not exist anymore
18
19 document.write (names[0] + "<br>"); // returns the 1st element of the array
20 names.shift(); // removes element from the end of the array
21 document.write (names[0] + "<br>"); // the 6th element of the array does not exist anymore
22 </script>

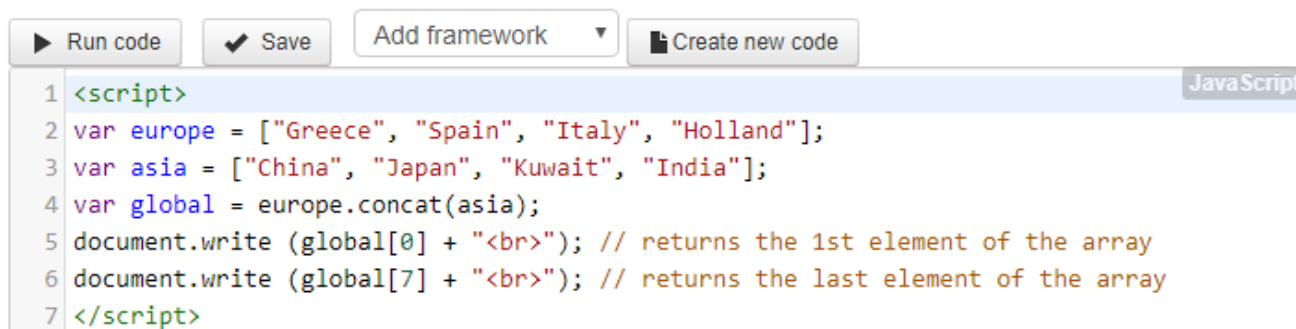
```

3. Run the code and see the results:

5
Clint
6
James
Stacey
Clint
Stacey
Stacey
Clint
undefined
Stacey
Nina

Example 2:

1. Go to <https://js.do/>
2. Type the code:



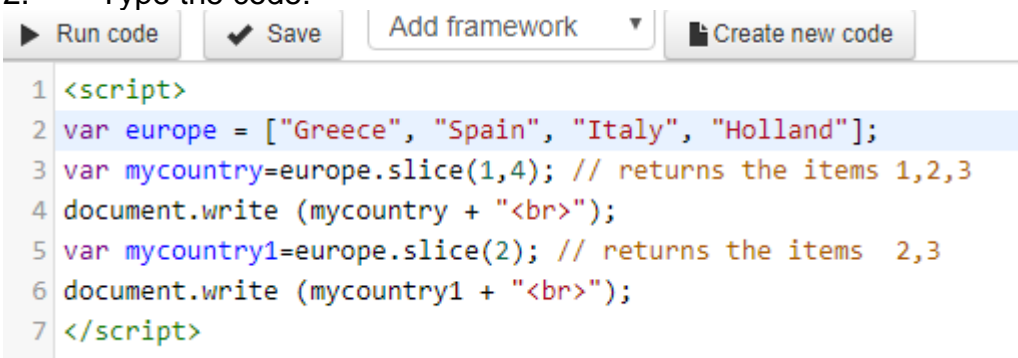
```
1 <script>
2 var europe = ["Greece", "Spain", "Italy", "Holland"];
3 var asia = ["China", "Japan", "Kuwait", "India"];
4 var global = europe.concat(asia);
5 document.write (global[0] + "<br>"); // returns the 1st element of the array
6 document.write (global[7] + "<br>"); // returns the last element of the array
7 </script>
```

3. Run the code and see the results:

Greece
India

Example 3:

1. Go to <https://js.do/>
2. Type the code:



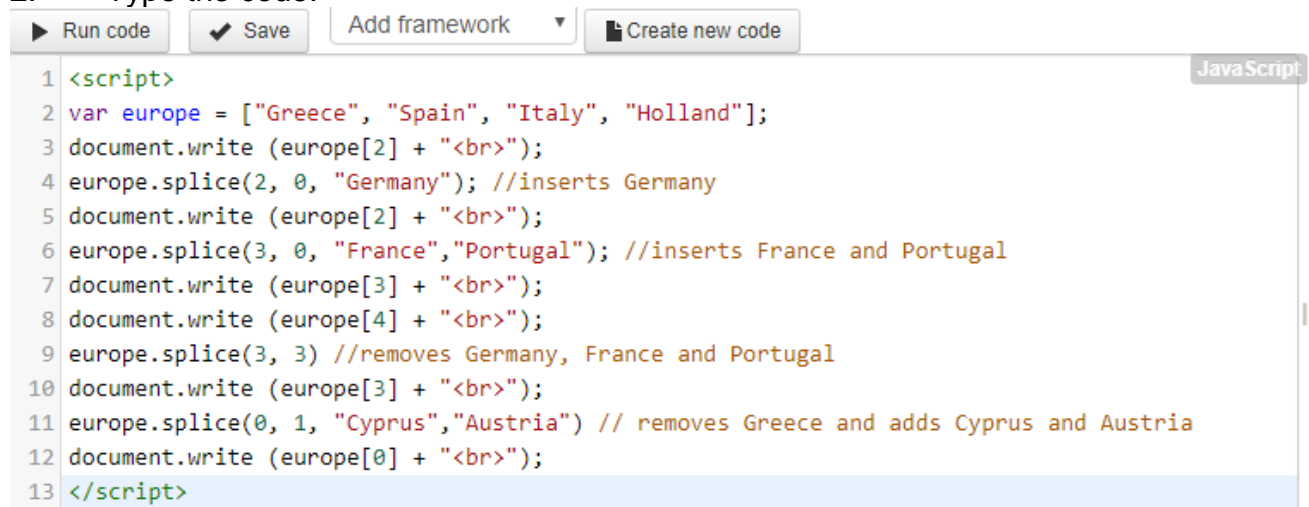
```
1 <script>
2 var europe = ["Greece", "Spain", "Italy", "Holland"];
3 var mycountry=europe.slice(1,4); // returns the items 1,2,3
4 document.write (mycountry + "<br>");
5 var mycountry1=europe.slice(2); // returns the items 2,3
6 document.write (mycountry1 + "<br>");
7 </script>
```

3. Run the code and see the results

Spain,Italy,Holland
Italy,Holland

Example 4:

1. Go to <https://js.do/>
2. Type the code:



```
1 <script>
2 var europe = ["Greece", "Spain", "Italy", "Holland"];
3 document.write (europe[2] + "<br>");
4 europe.splice(2, 0, "Germany"); //inserts Germany
5 document.write (europe[2] + "<br>");
6 europe.splice(3, 0, "France","Portugal"); //inserts France and Portugal
7 document.write (europe[3] + "<br>");
8 document.write (europe[4] + "<br>");
9 europe.splice(3, 3) //removes Germany, France and Portugal
10 document.write (europe[3] + "<br>");
11 europe.splice(0, 1, "Cyprus","Austria") // removes Greece and adds Cyprus and Austria
12 document.write (europe[0] + "<br>");
13 </script>
```

3. Run the code and see the results

Italy
Germany
France
Portugal
Holland
Cyprus

8.3 REVIEW EXERCISE

1. An array variable can hold:
 - a. Just a single number between -128 and 128.
 - b. A video file in the format normally used for display on a computer.
 - c. An audio file in the format normally used for ring tones.
 - d. A list of student names that can be sorted using the function `sort()`.

2. What is the JavaScript data type of each of the following values?
 - a. `37`
 - b. `"Sarah"`
 - c. `["David", "Sarah", "Ann", "Bill"]`
 - d. `['Red', 'Green', 'Blue']`
 - e. `[100, 20, 25]`

LESSON 9 – ENHANCE YOUR CODE

After completing this lesson, you should be able to:

- Describe the characteristics of well-structured and documented code like: indentation, appropriate comments, descriptive naming
- Define the programming construct term comment. Outline the purpose of a comment in a program
- Use comments in a program

9.1 READABLE CODE



Concepts

Imagine you have just completed a program for a game you have invented. You know exactly how the code works because you have just written it. You save your work and move on to another project. Now imagine returning to the program after a few months; how quickly would you be able to familiarise yourself with the code? How easy would it be for someone else who has never seen it to figure it out?

In reality JavaScript programs can be quite large and contain many instructions. They are often worked on by more than one programmer at different stages. It is essential that a program's code is 'readable' i.e. the reader can easily understand what the program does and why.

The same general techniques for making code easier to understand apply to all programming languages, they are:

- Use of comments
- Organisation of code
- Descriptive names

9.2 COMMENTS



Concepts

A **comment** is a piece of text explaining what some part of the code does. It is a short description of a piece of code that humans can read but the computer will ignore, designed to help programmers understand what is happening within a program. Appropriate comments are a characteristic of well-structured and documented code. Comments on the code should help the author and other people understand what each section of code is doing.

Comments start with two slashes, //, and go on to the end of the line. Any text between // and the end of the line will be ignored by JavaScript

For a block of comments, use the /* */ symbols

// This is a comment

/* This is a block of comments

with many lines

*/

9.3 ORGANISATION OF CODE

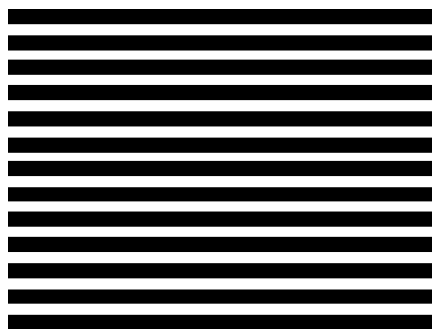


Concepts

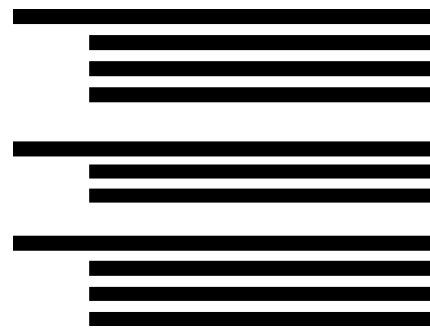
Programmers make their code easier to follow by breaking it up into smaller chunks, or blocks.

A block is defined as a group of adjacent lines that are indented the same amount. Indentation is used to make program code easier to read and understand. Many code editors perform the indentation automatically.

By using carefully named blocks, a programmer can give a clearer structure to a large JavaScript program.



'Unblocked' code



'Blocked' organisation of code

Figure 1

9.4 DESCRIPTIVE NAMES



Concepts

Descriptive naming is a characteristic of well-structured and documented code. It is the process of giving meaningful names to items and functions and procedures in programming. There is an accepted structure which is common to all languages. For example to write a small function called 'egg timer' in JavaScript, it should be written 'egg_timer'. This allows readers to see what the function is intended to do.

Giving meaningful to variable names will also make more sense to the reader. Variable names should be carefully selected so that they give a clue as to what the variable is used for.

In our list example from lesson 8, the variable name **pets** could have been used instead for a list of fruit:

```
pets=["pineapple", "mango", "kiwi fruit"]
```

JavaScript would have no problem with that, and if instructed:

```
document.write( pets[1] )
```

would print:

mango

However, using such a misleading name creates confusion for a programmer trying to read the code.

Example: Making Complex Code More Readable

1. Have a look at JavaScript program shown below. It is a complex piece of code, used here to illustrate some of the techniques discussed to enhance the readability of code.
2. See if you can identify the following:
 - a) Comments to explain the program
 - b) Named blocks of code

```
1 <script>
2
3 document.write("<b>LESSON 1: PRINT HELLO WORLD USING VARIABLES</b><br><br>");
4
5 // To start we will create 2 variables:
6 var variable1 = 'Hello ';
7 var variable2 = 'World!<br><br>'; // <br> means line break in html
8
9 // Create a third variable by adding both variables:
10 var final_text = variable1 + variable2;
11
12 // Print the final result:
13 document.write(final_text);
14
15 document.write("<b>LESSON 2: SOLVE AN EQUATION</b><br><br>");
16
17 var n = 9;
18 var x = n*n;
19 document.write("Result of n*n=" + x);
20
21 document.write("<br><br><b>LESSON 3: PRINT NUMBERS FROM 1 TO 10</b><br><br>");
22
23 // Create a loop of 10 elements.
24 // Variable "i" starts with value 1 and while i<=10 it will increment 1 (i=i+1)
25 for (var i=1; i<=10; i=i+1) {
26
27     document.write(i); // Print the current "i" number
28
29     // Print a comma followed by a space if i < 10
30     if (i<10) {
31         document.write(", ");
32     }
33 }
```

3. Review the code you have created so far in ICDL Computing and see where you might improve its readability by using the techniques from this lesson.

9.5 REVIEW EXERCISE

1. A 'comment' in a computer program is:
 - a. Anything contained in quotes
 - b. Anything in the code that is before a procedure
 - c. The ASCII characters //
 - d. Text to help document the program.
2. You should use a comment:
 - a. After every single line in a program.
 - b. Before every single line in a program.
 - c. To make it easier for someone else to understand your program.
 - d. Only for the most difficult function in a program.
3. Variable names should be:
 - a. Short to make the program as fast as possible.
 - b. At least eight letters long, so that other people won't be able to guess them.
 - c. Made from letters and numbers, not just letters.
 - d. Chosen to make your program more easily understood.
4. JavaScript blocks of code are indicated by:
 - a. Surrounding them with quotes.
 - b. Starting them with '{' and ending them with '}'.
 - c. Starting them with '(' and ending them with ')'.
d. Indenting the block of code.

LESSON 10 – CONDITIONAL STATEMENTS

After completing this lesson, you should be able to:

- Define the programming construct term conditional statement. Outline the purpose of conditional statements in a program
- Define the programming construct term logic test. Outline the purpose of a logic test in a program
- Use Boolean logic expressions in a program
- Use IF...THEN...ELSE conditional statements in a program

10.1 SEQUENCE AND STATEMENTS



Concepts

In early computer languages each line of code of a program was one instruction and was complete in itself. Each instruction was followed in order by the computer, one after the other. This is called **sequential programming**.

In **sequential programming** instructions are executed, in order, one after another.

In computer languages a **statement** is the smallest standalone unit of code that is complete in itself.

An example of a statement is the `document.write ()` statement:

```
document.write ( "Something to print")
```

When programmers use the term “statement” they are referring to small components of code like `document.write ()`.

10.2 IF STATEMENT



Concepts

In a flowchart, a decision box can be considered a complete standalone unit. The decision box has a True or False outcome that allows a choice to be made on what the code executes next.

A **Conditional Statement** is used to evaluate an expression as True or False. The outcome, a True or False value, determines what is done next.

In JavaScript, conditional statements use a structured format and layout, and are written with the keyword **if** followed by a boolean expression to evaluate the condition, and then a colon.

For example:

```
var Age = prompt ("How old are you?")
```

```
if (Age ) < 14
```

```
document.write ( "You are younger than me" );
```

The indented block of code on the next line (the `document.write ()` command in this case) will be obeyed if the expression is evaluated as true.

The boolean expression (Age < 14) in the if statement is called a **logic test**.

A **Logic test** is an expression that gives a Yes or No, True or False answer. The answer affects what code runs next.

This is the structured format to use:

```
If expression  
    statement if true
```

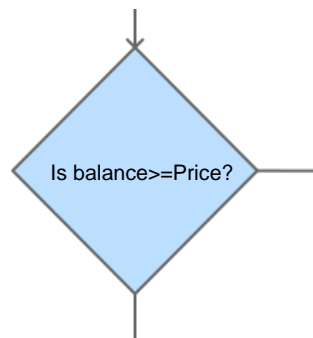
Syntax:

```
if (condition) {  
    // code to be executed if the condition is true  
}
```

In pseudocode the logic test may be expressed in natural language such as:

‘Do I have enough Money?’

The same logic test would be shown as text in a flowchart like this:



Example: A Logic Test

1. Go to <https://js.do/>
2. Type the code:

```
1 <script>  
2 var Name=prompt("What's your name?")  
3 if (Name.length <4) {  
4     document.write("You have a short name");  
5 }  
6 </script>
```

3. Run the program.
4. Respond to the prompt by typing either a short or a long name (more or fewer than 4 letters).
5. Observe the result.

You have a short name

10.3 IF...ELSE STATEMENT



Concepts

JavaScript can allow for one block of indented code, a subroutine, to run if the logic test is True, and different block of code to run if the logic test is False.

The code is written with the if statement as before, and a new keyword **else**, followed by a colon, to add code that runs if the logic test is False.

This is the structure format to use:

```
if expression:
    statement if true
else:
    statement if false
```

Syntax:

```
if (condition) {
    // code to be executed if the condition is true
} else {
    // of code to be executed if the condition is false
}
```

For example:

```
var Age = prompt ("How old are you?")
if (Age < 24 )
    document.write ( "You are younger than me" )
else
    document.write ( "You are older than me" )
```

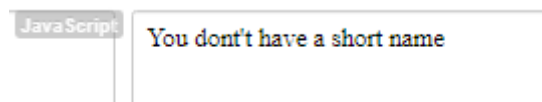
The indented block of code on the next line (the print() command in this case) will be obeyed if the expression is evaluated as true.

Example: An if..else Statement.

1. Go to <https://js.do/>
2. Type the code:

```
1 <script>
2 var Name=prompt("What's your name?")
3 if (Name.length <4) {
4     document.write("You have a short name");
5 } else {
6     document.write("You don't have a short name");
7 }
8 </script>
```

3. Run the code.
4. Type in a short name.
5. Run the code again.
6. Type in a longer name to see the code running for the case where the logic test is false.



10.4 SWITCH STATEMENT



Concepts

The switch statement is used to perform different actions based on different conditions.

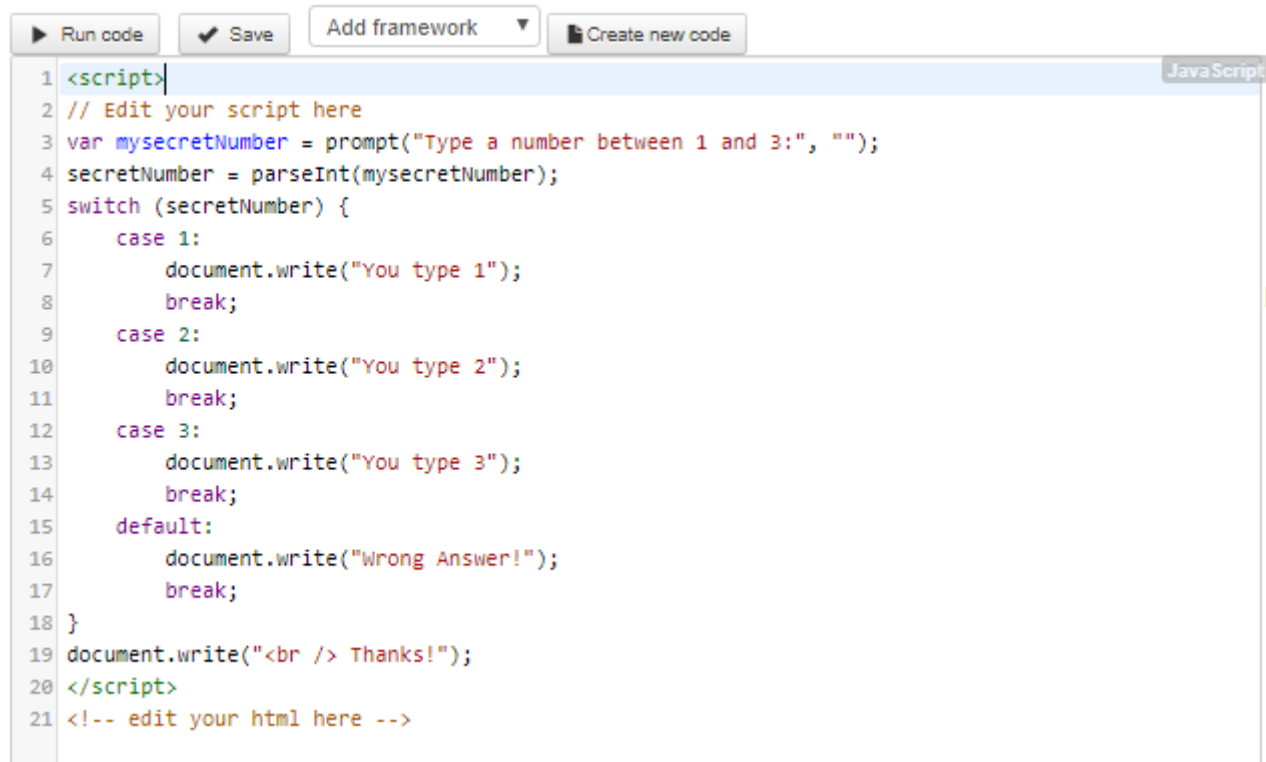
You saw earlier how the if and else if statements could be used for checking various conditions; if the first condition is not valid, then another is checked, and another, and so on. However, when you want to check the value of a particular variable for a large number of possible values, there is a more efficient alternative, namely the switch statement.

This is the structure format to use:

```
switch (expression) {  
    case value1:  
        statement;  
        break;  
    case value2:  
        statement;  
        break;  
    case value3:  
        statement;  
        break;  
    default:  
        statement;  
}
```

Example: Switch Statement.

1. Go to <https://js.do/>
2. Type the code:



```
1 <script>
2 // Edit your script here
3 var mysecretNumber = prompt("Type a number between 1 and 3:", "");
4 secretNumber = parseInt(mysecretNumber);
5 switch (secretNumber) {
6     case 1:
7         document.write("You type 1");
8         break;
9     case 2:
10        document.write("You type 2");
11        break;
12    case 3:
13        document.write("You type 3");
14        break;
15    default:
16        document.write("Wrong Answer!");
17        break;
18 }
19 document.write("<br /> Thanks!");
20 </script>
21 <!-- edit your html here -->
```

3. Run the code.
4. Type 1 or 2 or 3.
5. Run the code again.
6. Type a number different than 1,2 or 3.

Wrong Answer!
Thanks!

Note: the `parseInt()` function to converts the string that is returned from `prompt()` to an integer value.

10.5 REVIEW EXERCISE

1. A JavaScript conditional statement:
 - a. Has the keyword 'if' in it
 - b. Has the keyword 'def' in it
 - c. Is a special kind of comment saying what conditions the code should be used in.
 - d. Is a special kind of procedure that may in some conditions be a function instead.

2. In the following code:

```
if password = "Voldemort" :  
    document.write( "I guessed your password correctly" );  
else :  
    document.write( "I did not guess your password correctly");
```

...which of the following statements are correct?

- a. Both the first and second indented block will always be run.
- b. Neither the first nor the second indented block will ever be run
- c. There is a logic error because you cannot use == with a string.
- d. The first or second indented block may be run. Which one runs depends on the value of password.

LESSON 11 – FUNCTIONS

After completing this lesson, you should be able to:

- Understand the term function. Outline the purpose of a function in a program
- Write and name a function in a program

11.1 FUNCTION



Concepts

One piece of code may be used more than once in different places in a program. This is called a function.

A **Function** is a block of code that can be called up and run many times in a program. For example a function could be used to display the date or time on the computer screen.

The function will run and when it has finished, the main program resumes from before the start of the function.

A function can perform a calculation, and return the answer or value to the program. This is called **returning** a value.

A function contains JavaScript statements that behaves as a single command and can be recalled repeatedly.

11.2 FUNCTIONS SYNTAX AND INVOCATION



Concepts

A function is a way of writing code once but using it many times. Functions help the user to avoid repetition and make the code more organized and easier to update and maintain.

A JavaScript function is defined with the function keyword, parentheses () and the code to be executed, by the function, is placed inside brackets: {}

```
function name() {  
    // code to be executed  
}
```

Once a function is defined, we can assign it to a variable, just like any value.

The code inside a JavaScript function will execute when something invokes it. Simply call the function by name in parenthesis.

Example:

1. Go to <https://js.do/>
2. Type the code:

```
1 <script>
2 var myName; // declare variable
3 myName = function () { // define function and assign function
4 document.write("My name is John"); // code to be executed
5 };
6 myName(); // call function
7 </script>
```

3. Run the code

Functions, Parameters – Arguments

Function parameters are listed inside the parentheses () in the function definition.

Function arguments are the values received by the function when it is invoked.

Inside the function, the arguments (the parameters) behave as local variables.

```
function name() { parameter1, parameter2, parameter3
    // code to be executed
}
```

Example:

1. Go to <https://js.do/>
2. Type the code:

```
1 <script>
2 var mySum; // declare variable
3 mySum = function (num1, num2) { // define function with 2 parameters and assign function
4 var mytotal = num1 + num2;
5 document.write("My sum is " + mytotal); // code to be executed
6 };
7 mySum(5, 3); // call function
8 document.write("<br>");
9 mySum(2, -4); // call function
10 </script>
```

3. Run the code:

When we call the mySum function, the program automatically assigns the two arguments to the two parameters in the definition, num1 and num2.

Returning a value from a function

It's useful to have a function do some work and give us back the result of that work. Return a value from a function by using the **return** keyword. Whatever follows return in a statement is the value that replaces the function call.

Example:

1. Go to <https://js.do/>
2. Type the code

```
1 <script>
2 var myMessage;
3 myMessage = function () {
4   return "Have a nice day!"; //return the message from the function
5 };
6 document.write (myMessage());
7 </script>
```

3. Run the code:

The myMessage function always returns the same value.

Example:

1. Go to <https://js.do/>
2. Type the code

```
1 <script>
2 var mySum;
3 mySum = function (num1, num2) {
4   return num1 + num2; //return keyword to pass the result
5 };
6 var result = mySum(50, 23); //Call the mySum function and assign the value
7 document.write(result);
8 </script>
```

3. Run the code:

mySum(50, 23) calls the mySum function, assigning 50 to num1 and 23 to num2. The value of number1 is then added to the value of number2, the return keyword returns the addition and then ends the function.

var result = mySum(50, 23);

becomes

var result = 73;

because mySum(50, 23) returns 73.

11.3 REVIEW EXERCISE

1. A typical function in JavaScript:
 - a. Starts with the keyword def and provides some code that returns a value
 - b. Starts with the keyword function and provides some code that returns a value
 - c. Is a named piece of code that does a number of things and then stops the program.
 - d. Is a named piece of code that does a number of things and then restarts the JavaScript shell.
2. A typical function returns a value by using the _____ keyword in JavaScript:
 - a. ...def...
 - b. ...return...
 - c. ...#...
 - d. ...var..
3. A parameter is:
 - a. A value which is passed in to a function for it to use
 - b. A way of measuring distance approximately, that can makes some code simpler.
 - c. A way for a program to do several things at the same time.
 - d. A block of code with a specified number of lines to it.

LESSON 12 – LOOPS

After completing this lesson, you should be able to:

- Define the programming construct term loop. Outline the purpose and benefit of looping in a program
- Recognise types of loops used for iteration: for, while
- Understand the term infinite loop
- Use iteration (looping) in a program like: for, while

12.1 LOOPING

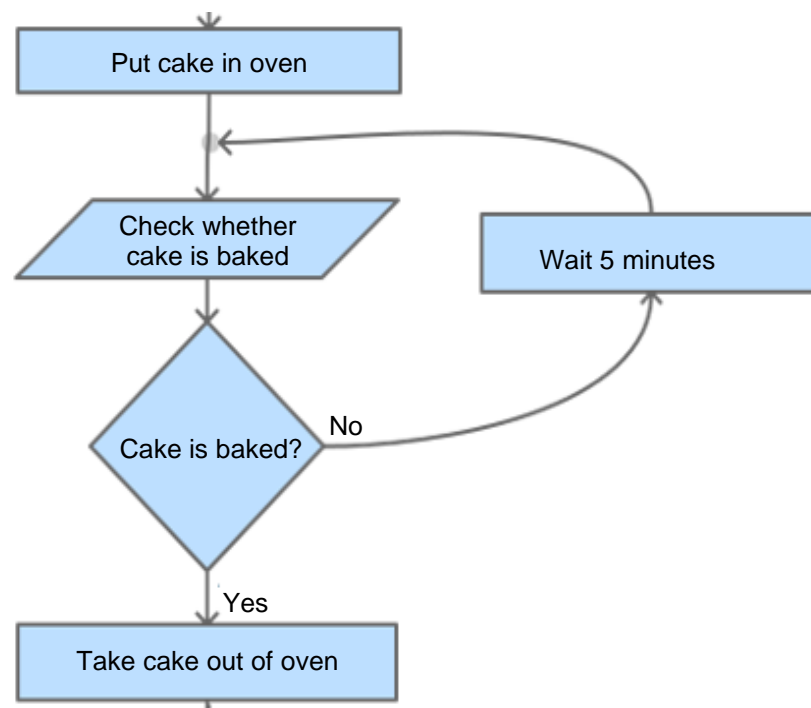


Concepts

Up to this point we have created programs that work sequentially, i.e. statements are executed in the order they are written, like following a recipe step by step. If we need to carry out the same step several times, we can use what's known as a **loop** in computing.

A **loop** is a piece of code that is run repeatedly under certain conditions.

A loop is represented in flowcharts where an arrow from a decision box goes back to an earlier box in the sequence. Examine the flowchart for baking the cake below. Checking the cake to see if it is baked is a step that could be repeated several times before the cake is ready to be taken out of the oven. This repeatable step is represented by a loop – check cake, if not baked, wait 5 minutes then check again.



A loop in a flowchart

Using a loop to repeat an action is one of the most useful techniques in coding. Looping code is present in most programming languages with slight variations on naming conventions e.g. 'repeat' is used to determine how many times to loop in some languages; we'll see later in this lesson that 'while' does the same thing in JavaScript.

JavaScript uses the '**for**' statement to make loops.

The **for** statement has several parts to it.

```
for (i = 0; i < 10; i++) {  
    document.write (i);  
}
```

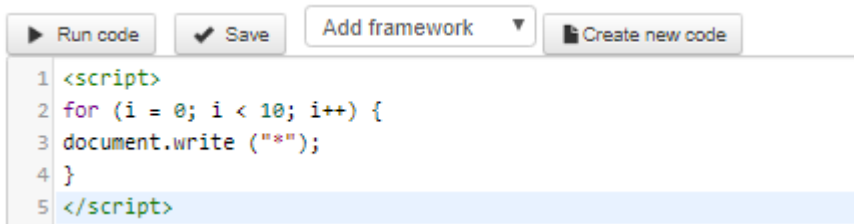
- The keyword 'for' is followed by the name of a variable, in this case i, keeping count of the repeats of the loop. The variable i will have an initial value 0, and after each time the loop is run, it increments to 1 then 2, and so on up to 9.
- The words 'in range()' with a number inside the parentheses followed by a colon, indicating how many times the loop will run
- An indented line of code with a statement will be executed 10 times.

Result: **0123456789**

Example: Attempting to Print a Row of Ten Stars

This example shows an attempt to print a row of ten stars.

1. Go to <https://js.do/>
2. Type the code



```
1 <script>  
2 for (i = 0; i < 10; i++) {  
3 document.write ("*");  
4 }  
5 </script>
```

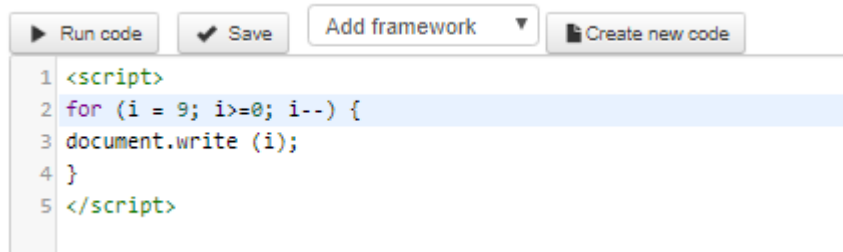
3. Run the code.



```
*****
```

Example: Going Backward

1. Go to <https://js.do/>
2. Type the code



```
1 <script>
2 for (i = 9; i>=0; i--) {
3   document.write (i);
4 }
5 </script>
```

3. Run the code.

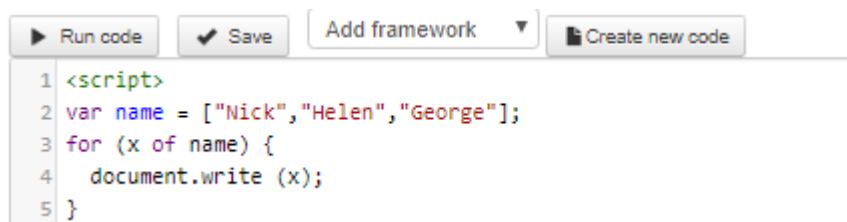
Result: **9876543210**

The **for...of** loops through the values of an array.

```
for (variable of array) {
  // code block to be executed
}
```

Example:

1. Go to <https://js.do/>
2. Type the code



```
1 <script>
2 var name = ["Nick", "Helen", "George"];
3 for (x of name) {
4   document.write (x);
5 }
```

3. Run the code

Result: **Nick,Helen,George**

12.2 LOOPING WITH VARIABLES



Concepts

In the previous example each star was displayed on a line of its own. We wanted a row of stars.

One different approach would be:

```
document.write("*****")
```

This will work but it is not efficient – what if you wanted to print 599 stars?

This is where loops save a lot of programming time and lead to shorter code. We can write a loop to print 10 stars, 42 stars or 599 stars, depending on a value.

How the program works:

- Print a row of 10 stars by using a loop is to build up the row of stars in a variable. Then, once the row is ready, the code can print it. This approach will use the + operator to add the string "*" onto a variable called answer.

```
answer = answer + "*"
```

This line of code uses the current value of answer, adds a star onto the end, and puts the result back into answer.

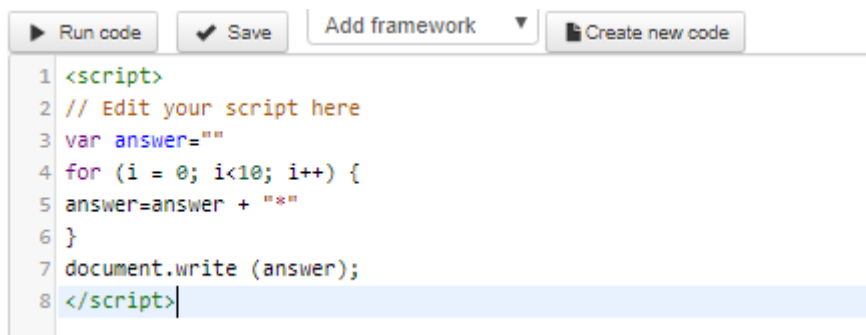
- The variable named answer must first be initialised before it is used. To initialise it to a string with three stars use:

```
answer = "***"
```

To initialise answer to a string that is empty, remove the three stars from the line above. Keep the two quotation marks! Without them it's an error. The two quotation marks at the start and end tell JavaScript that answer is a string.

Example: Row of Stars using a Variable

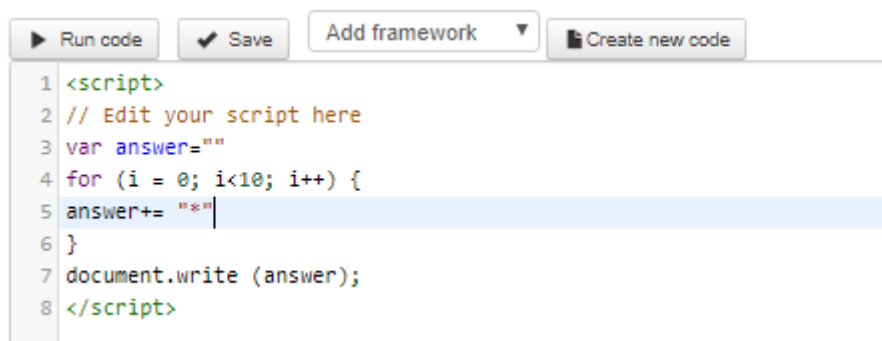
1. Go to <https://js.do/>
2. Type the code
3. Initialise the variable called answer to be an empty string.
4. Use a loop to add 10 stars onto the string.
5. Print the string.



```
1 <script>
2 // Edit your script here
3 var answer=""
4 for (i = 0; i<10; i++) {
5   answer=answer + "*"
6 }
7 document.write (answer);
8 </script>
```

There is a shorthand way of writing `answer=answer+"*"` which uses a new operator `+=`.

6. Write the same code using the `+=` operator, as shown below:



```
1 <script>
2 // Edit your script here
3 var answer=""
4 for (i = 0; i<10; i++) {
5   answer+= "*"
6 }
7 document.write (answer);
8 </script>
```

Result: *****

12.3 VARIATIONS ON LOOPS

Concepts

We have seen that loops have a particular structure and layout in JavaScript.

```
for (statement 1; statement 2; statement 3) {
  // code block to be executed
}
```

The indented code block after the 'for' line is said to be 'inside the loop'.

- Code before the loop is executed and then the loop is executed.
- Code inside the loop is executed multiple times.
- Code immediately after the loop is executed only after the loop has finished repeating the code in the indented block.

So far we have looked at the basic loop but there are other variations of loops that we can use as well.

Another kind of loop uses a logic test rather than `range()`. It's called a while loop.

A **while loop** tests a boolean expression and repeats execution of the loop as long as (ie While) the boolean expression is true.

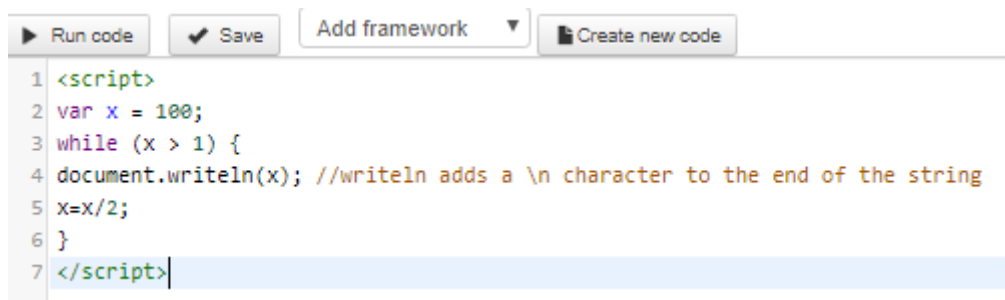
```
while (condition) {  
    // code block to be executed  
}
```

How the program works:

- The example that follows will initialise a variable x to 100 and keep dividing by 2 while x is greater than 1.
- Then it will stop.

Example: A While Loop

1. Go to <https://js.do/>
2. Type the code
3. Type in the code as shown below:



```
1 <script>  
2 var x = 100;  
3 while (x > 1) {  
4 document.writeln(x); //writeln adds a \n character to the end of the string  
5 x=x/2;  
6 }  
7 </script>
```

4. Run the code.

```
100 50 25 12.5 6.25 3.125 1.5625
```

In a while loop, if the logic test is always true, the loop will repeat forever. It then becomes an infinite loop.

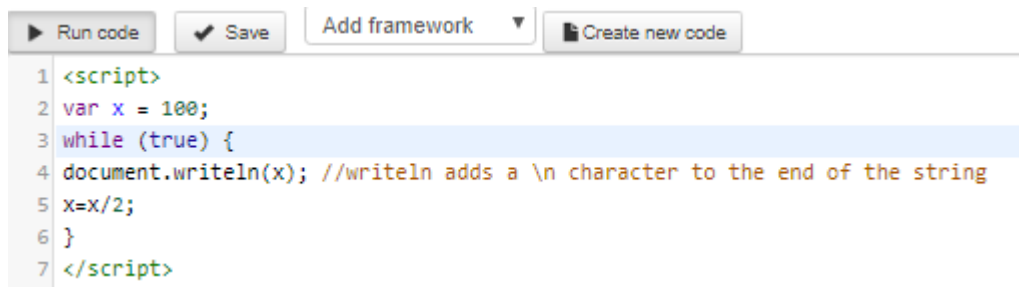
An infinite loop never stops. It uses the key words "while true"

As we usually want a program to stop at some point, infinite loops usually indicate an error.

Example: An Infinite Loop

1. Go to <https://js.do/>
2. Type the code

3. Type in the code as shown below:



```
1 <script>
2 var x = 100;
3 while (true) {
4 document.writeln(x); //writeln adds a \n character to the end of the string
5 x=x/2;
6 }
7 </script>
```

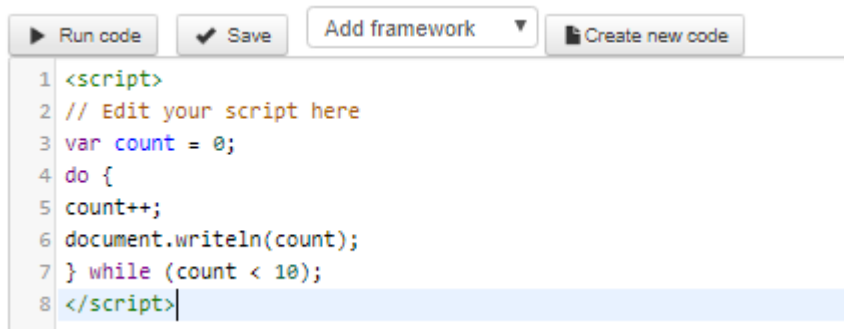
4. Run the program.

5. You can stop the program by closing the tab.

The **do/while** loop will execute the code block once, before checking if the condition is true, then it will repeat the loop as long as the condition is true.

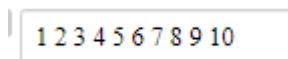
```
do {
    // code block to be executed
}
while (condition);
```

1. Go to <https://js.do/>
2. Type the code
3. Type in the code as shown below:



```
1 <script>
2 // Edit your script here
3 var count = 0;
4 do {
5 count++;
6 document.writeln(count);
7 } while (count < 10);
8 </script>
```

4. Run the program.



```
1 2 3 4 5 6 7 8 9 10
```

12.4 REVIEW EXERCISE

1. What is the main purpose of a loop?
 - a. To stop the computer running off the end of a program.
 - b. To go back to the beginning of a program.
 - c. To connect one computer to another.
 - d. To do some instructions repeatedly.
2. A loop that goes on forever is called?
 - a. A pixel loop.
 - b. A broken loop.
 - c. An infinite loop.
 - d. An 'iffy-loop'.
3. To count through several items, we typically use:
 - a. a loop the loop
 - b. random numbers
 - c. a while loop
 - d. a for loop
4. To repeat some action several times without counting we typically use:
 - a. a loop the loop
 - b. random numbers
 - c. a while loop
 - d. a for loop

LESSON 13 – OBJECTS - EVENTS

After completing this lesson, you should be able to:

- Perform many common mathematical calculations
- Use string methods to manipulate the strings in a variety of ways
- Use date object find out the current date and time, store your own dates and times, do calculations with these dates, and convert the dates into strings
- Use event handlers like: mouse click, keyboard input, button click, timer

13.1 MATH OBJECT



Concepts

Math Object

The Math object's methods allow you to perform many common mathematical calculations. An object's methods are called by writing the name of the object followed by a dot (.) and the name of the method. In parentheses following the method name is the argument to the method.

For example, to calculate and display the square root of 25 you might write:

document.write(Math.sqrt(25)); which calls method **Math.sqrt** to calculate the square root of the number contained in the parentheses (25), then outputs the result. The number 25 is the argument of the Math.sqrt method.

METHOD	DESCRIPTION	EXAMPLE
abs(x)	absolute value of x	abs(7.2) is 7.2 abs(0.0) is 0.0 abs(-5.6) is 5.6
ceil(x)	rounds x to the smallest integer not less than x	ceil(9.2) is 10.0 ceil(-9.8) is -9.0
cos(x)	trigonometric cosine of x (x in radians)	cos(0.0) is 1.0
exp(x)	exponential method e^x	exp(1.0) is 2.71828 exp(2.0) is 7.38906
floor(x)	rounds x to the largest integer not greater than x	floor(9.2) is 9.0 floor(-9.8) is -10.0
log(x)	natural logarithm of x (base e)	log(2.718282) is 1.0 log(7.389056) is 2.0
max(x, y)	larger value of x and y	max(2.3, 12.7) is 12.7 max(-2.3, -12.7) is -2.3
min(x, y)	smaller value of x and y	min(2.3, 12.7) is 2.3 min(-2.3, -12.7) is -12.7
pow(x, y)	x raised to power y (x^y)	pow(2.0, 7.0) is 128.0 pow(9.0, 0.5) is 3.0

round(x) (x in radians)	rounds x to the closest integer	round(9.75) is 10 round(9.25) is 9
sin(x)	trigonometric tangent of x (x in radians)	sin(0.0) is 0.0
sqrt(x)	square root of x	sqrt(900.0) is 30.0 sqrt(9.0) is 3.0
tan(x)	trigonometric sine of x	tan(0.0) is 0.0

Example Math Methods:

1. Go to <https://js.do/>
2. Type the code:

```

1 <script>
2 document.write("sq= " + Math.sqrt(25));
3 document.write ("<br>")
4 document.write("min= " + Math.min(30,4) );
5 document.write ("<br>")
6 document.write("pow= " + Math.pow(2,3));
7 </script>

```

3. Run the code:

Results: sq= 5 min= 4 pow= 8

13.2 STRING OBJECT



Concepts

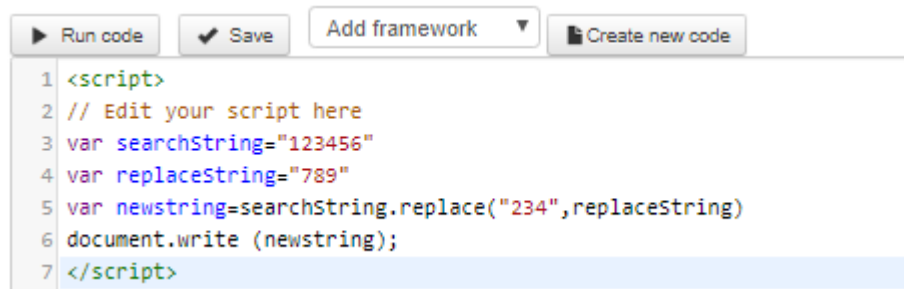
A string is a series of characters treated as a single unit. The first character is in position 0, the second in 1, and so on. Strings can be compared via the relational (<, <=, > and >=) and equality operators (== and !=).

METHOD	DESCRIPTION
replace(searchString, replaceString)	searches a string for a specified value and returns a new string where the specified values are replaced
substr(start, length)	extracts parts of a string, beginning at the character at the specified position, and returns

	the specified number of characters
substring(start, end)	extracts the characters from a string, between two specified indices, and returns the new sub string
toLowerCase()	converts a string to lowercase letters
toUpperCase()	converts a string to uppercase letters
lastIndexOf(substring, index)	eturns the position of the last occurrence of a specified value in a string
indexOf(substring, index)	returns the position of the first occurrence of a specified value in a string

Example1 String Methods:

1. Go to <https://js.do/>
2. Type the code:



```

1 <script>
2 // Edit your script here
3 var searchString="123456"
4 var replaceString="789"
5 var newString=searchString.replace("234",replaceString)
6 document.write (newString);
7 </script>

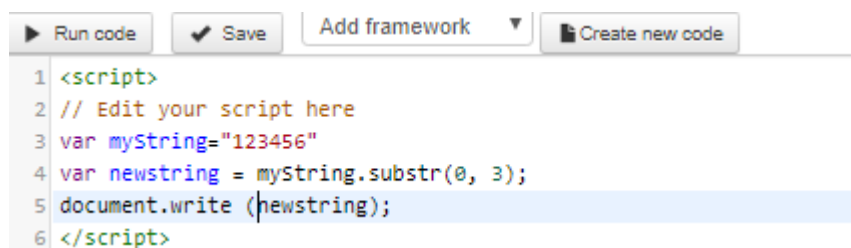
```

3. Run the code:

Result: 178956

Example2 String Methods:

1. Go to <https://js.do/>
2. Type the code:



```

1 <script>
2 // Edit your script here
3 var myString="123456"
4 var newString = myString.substr(0, 3);
5 document.write (newString);
6 </script>

```

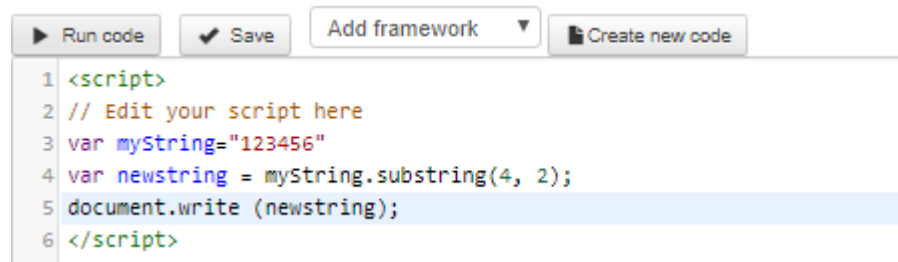
3. Run the code:

Result: 123

Example3 String Methods:

1. Go to <https://js.do/>

2. Type the code:

A screenshot of a web-based code editor. At the top, there are four buttons: 'Run code' (with a play icon), 'Save' (with a checkmark icon), 'Add framework' (with a dropdown arrow), and 'Create new code' (with a document icon). Below the buttons is a text area containing six lines of JavaScript code. Line 1: <script>. Line 2: // Edit your script here. Line 3: var myString="123456". Line 4: var newstring = myString.substring(4, 2);. Line 5: document.write (newstring);. Line 6: </script>. The line numbers 1 through 6 are in the left margin. The code is color-coded: <script> and </script> are green, // is orange, var is blue, myString and newstring are blue, "123456" is red, = is black, myString.substring(4, 2); is blue, document.write is blue, and (newstring); is blue. The line document.write (newstring); is highlighted with a light blue background.

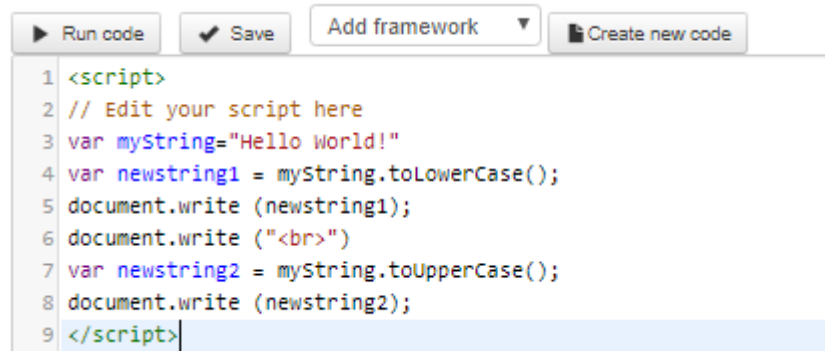
```
1 <script>
2 // Edit your script here
3 var myString="123456"
4 var newstring = myString.substring(4, 2);
5 document.write (newstring);
6 </script>
```

4. Run the code:

Result: 34

Example4 String Methods:

1. Go to <https://js.do/>
2. Type the code:



```
1 <script>
2 // Edit your script here
3 var myString="Hello World!"
4 var newstring1 = myString.toLowerCase();
5 document.write (newstring1);
6 document.write ("<br>")
7 var newstring2 = myString.toUpperCase();
8 document.write (newstring2);
9 </script>
```

3. Run the code:

Result: hello world!
HELLO WORLD!

Example5 String Methods:

1. Go to <https://js.do/>
2. Type the code:



```
1 <script>
2 // Edit your script here
3 var myString="Hello World!"
4 var newstring1 = myString.lastIndexOf("o");//last occurrence of o
5 var newstring2 = myString.indexOf("o");//first occurrence of o
6 document.write (newstring1);
7 document.write ("<br>")
8 document.write (newstring2);
9 </script>
```

3. Run the code:

Result: 7
4

13.3 DATE OBJECT



Concepts

Using date object in JavaScript, you can find out the current date and time, store your own dates and times, do calculations with these dates, and convert the dates into strings.

Creating Date Objects

To create many types of objects, you use the **new** operator. The following statement creates a Date object: **var myDate = new Date();**

The **var** keyword to define a variable called **myDate**. This variable is initialized, using the equals sign assignment operator (=), to the right - hand side of the statement.

The right - hand side of the statement consists of two parts. First parts has the operator **new**. This tells JavaScript to create a new object. Next part has **Date()**. This is the constructor for a **Date** object. It's a function that tells JavaScript what type of object you want to create.

The constructor is a function, so we can pass parameters to the constructor to add data to the object. For example, the following code creates a Date object containing the date 11 May 2022:

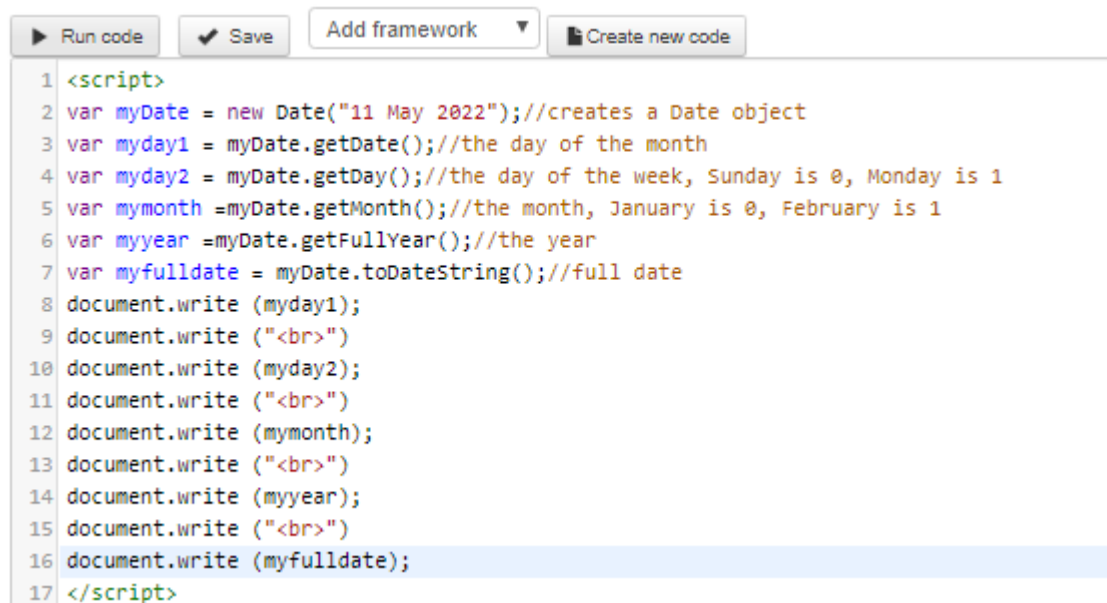
```
var myDate = new Date("11 May 2022");
```

The date object has a lot of methods:

METHOD	DESCRIPTION
getDate()	returns the day of the month
getDay	returns the day of the week as an integer
getMonth()	returns the month as an integer
getFullYear	returns the year as a four - digit number
toDateString	returns the full date based on the current time zone

Example Date Methods:

1. Go to <https://js.do/>
2. Type the code:



```

1 <script>
2 var myDate = new Date("11 May 2022");//creates a Date object
3 var myday1 = myDate.getDate();//the day of the month
4 var myday2 = myDate.getDay();//the day of the week, Sunday is 0, Monday is 1
5 var mymonth =myDate.getMonth();//the month, January is 0, February is 1
6 var myyear =myDate.getFullYear();//the year
7 var myfulldate = myDate.toDateString();//full date
8 document.write (myday1);
9 document.write ("<br>")
10 document.write (myday2);
11 document.write ("<br>")
12 document.write (mymonth);
13 document.write ("<br>")
14 document.write (myyear);
15 document.write ("<br>")
16 document.write (myfulldate);
17 </script>

```

3. Run the code:

11
3
4
2022
Wed May 11 2022

13.4 EVENTS



Concepts

The flow of a program can depend on certain actions happening. Users can trigger these actions by, for example, using the mouse to click on a graphic they see on screen, pressing a key on the keyboard or clicking a link in a browser window. In JavaScript programming these actions are called events.

Events are used extensively in games. Events that determine the movement of game's character are triggered by user input, for example: using the arrow keys to move the character in a particular direction, using a mouse click to make the character jump etc.

Some events may be specific to hardware. In an alarm system which uses a computer to monitor for intruders, a person moving near a sensor or pressure on a pressure mat could cause or trigger an event.

In some computers, the battery being low would trigger an event. This may cause a message to display to suggest recharging the battery.

Event Handlers

An Event handler is a piece of code designed to do something once an event has been triggered. For example the up, down, left, right arrows on a keyboard can control the movement of a player in a game. Depending on which event occurs (Up, down, left, right), the program will execute the appropriate 'event handler' code which will carry out the correct movement of the player.

Some computer languages have a system of 'registering' functions that are called when certain events occur. We can recognise these functions by their names which begin with the word 'On' followed by the name of the event. For example 'OnClick' is called when a user generates an event by clicking anywhere on the screen.

Example: Exploring Event Handlers

Events and JavaScript

Event	Description
onchange	An HTML element has been changed
onclick	The user clicks the mouse an HTML element
ondblclick	The user double clicks the mouse an HTML element
onmouseover	The user moves the mouse over an HTML element
onmouseout	The user moves the mouse away from an HTML element
onkeydown	The user presses down a keyboard key
onkeyup	The user releases a keyboard key
onload	The browser has finished loading the page

You can connect your code to an event in three ways:

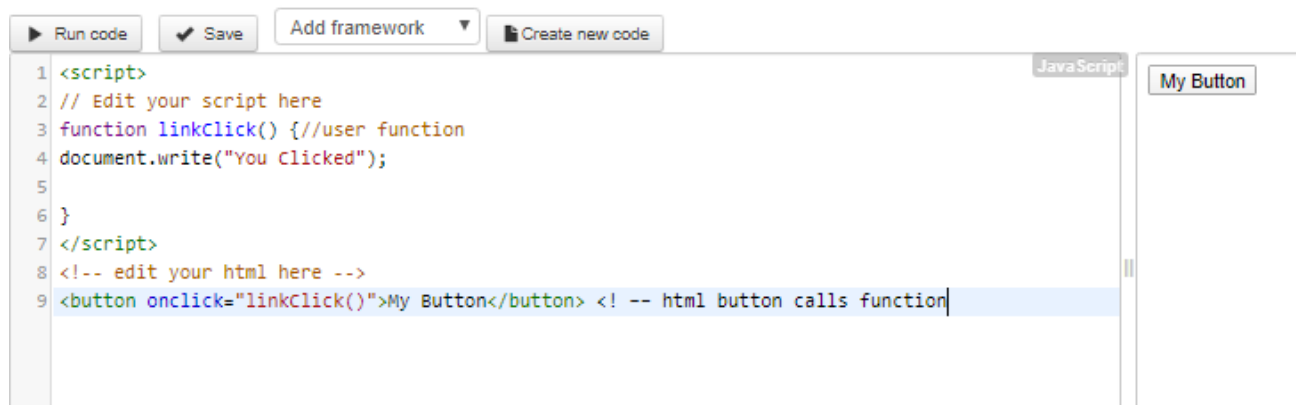
Assigning HTML attributes

Assigning an object's special properties

Calling an object's special methods

Example1

1. Go to <https://js.do/>
2. Type the code:



The screenshot shows the js.do online code editor interface. At the top, there are buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The main editor area contains the following code:

```
1 <script>
2 // Edit your script here
3 function linkClick() { //user function
4 document.write("You Clicked");
5
6 }
7 </script>
8 <!-- edit your html here -->
9 <button onclick="linkClick()">My Button</button> <!-- html button calls function
```

On the right side of the editor, there is a preview area showing a button labeled 'My Button'.

3. Run the code, click the button:



The screenshot shows a code editor interface with a top bar containing buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code editor displays the following JavaScript code:

```
1 <script>
2 // Edit your script here
3 function linkClick() { //user function
4 document.write("You Clicked");
5
6 }
7 </script>
8 <!-- edit your html here -->
9 <button onclick="linkClick()">My Button</button> <!-- html button calls function
```

On the right side of the editor, there is a 'JavaScript' tab and a preview area showing the output 'You Clicked'.

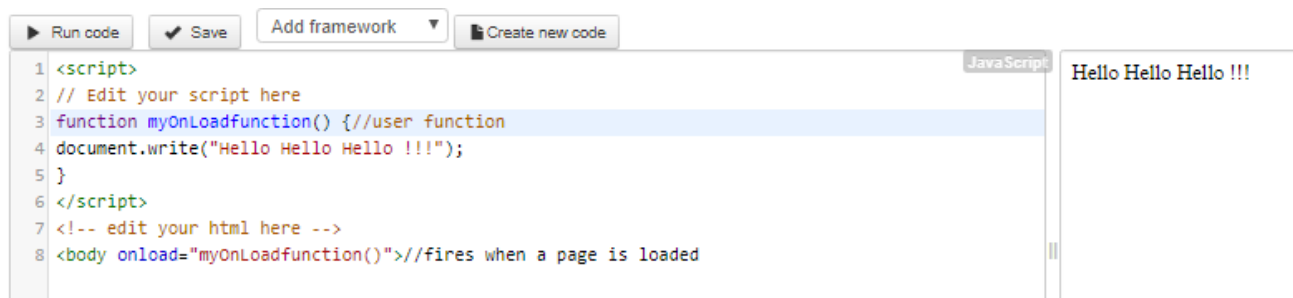
How the program works:

Within the script block, there is a standard function. The onclick attribute is now connected to some code that calls the function linkClick(). Therefore, when the user clicks the button, this function will be executed.

Some events are not directly linked with the user's actions. For example, the window object has the load event, which fires when a page is loaded.

Example2

1. Go to <https://js.do/>
2. Type the code:
3. Run the code, click the button:



The screenshot shows a code editor interface with a top bar containing buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code editor displays the following JavaScript code:

```
1 <script>
2 // Edit your script here
3 function myOnLoadfunction() { //user function
4 document.write("Hello Hello Hello !!!");
5 }
6 </script>
7 <!-- edit your html here -->
8 <body onload="myOnLoadfunction()"> //fires when a page is loaded
```

On the right side of the editor, there is a 'JavaScript' tab and a preview area showing the output 'Hello Hello Hello !!!'.

How the program works:

Within the script block, there is a standard function. The onload attribute is now connected to some code that calls the function myonLoadfunction(), which will be executed when the page is loaded.

Example2

1. Go to <https://js.do/>
2. Type the code:

```
<script>
// Edit your script here
function dblclick() { //user function
    document.getElementById("test").innerHTML = "Double clicked"; //The
    document.getElementById() method returns the element of specified id.
}
</script>
<!-- edit your html here -->
<p onclick="dblclick()">Doubleclick this paragraph!!!</p>
<p id="test"></p>
```

3. Run the code:

Result: Doubleclick this paragraph!!!

Double clicked

How the program works:

Within the script block, there is a standard function. The onclick attribute is now connected to some code that calls the function `dblclick()`, which will be executed when the user double click the paragraph.

13.7 REVIEW EXERCISE

1. An event is:
 - a. When a program is running and moves from code written by the programmer into library code.
 - b. Something that happens that a program should react to.
 - c. When a program has too much data and needs to get rid of some of the data.
 - d. A special JavaScript data type used to track important dates, for example in calendar apps.

2. The method floor(x) _____
 - a. ... squares root of x.
 - b. ... raises to power x.
 - c. ... rounds x.
 - d. ...counts the absolute value of x.

3. Which method returns the day of the month?
 - a. getDate()
 - b. newDate()
 - c. getDay()
 - d. getMonth()

4. Which of the following generates events in JavaScript?
 - a. Clicking on the mouse.
 - b. A timer.
 - c. Input from keyboard.
 - d. All the above.

LESSON 14 – RECURSION

After completing this lesson, you should be able to:

- Understand the term recursion

This lesson also reinforces concepts from earlier lessons such as:

- Loop
- Procedure
- Function
- Variable
- Parameter

14.1 RECURSION



Concepts

The programs we have discussed thus far are generally structured as functions that call one another in a disciplined, hierarchical manner. A recursive function is a function that calls itself, either directly, or indirectly through another function. In this section, we present a simple example of recursion..

A function that calls itself is known as Recursion.

Recursion is the process of a function dividing a problem into simpler parts and calling itself to solve those simpler parts. A recursive function has a call to itself within the function.

When a recursive function invokes itself it is said to **recurse**.

The technique of recursion is most useful when a problem can be decomposed into two smaller problems that are still in many ways like the original problem.

The recursion step executes while the original call to the function is still open (i.e., it has not finished executing). The recursion step can result in many more recursive calls as the function divides each new subproblem into two conceptual pieces. For the recursion eventually to terminate, each time the function calls itself with a simpler version of the original problem, the sequence of smaller and smaller problems must converge on the base case. At that point, the function recognizes the **base case**, returns a result to the previous copy of the function, and a sequence of returns ensues up the line until the original function call eventually returns the final result to the caller.

Recursive Factorial

The factorial of a nonnegative integer n , written $n!$ (and pronounced “ n factorial”), is the product $n \cdot (n - 1) \cdot (n - 2) \cdot \dots \cdot 1$ where $1!$ is equal to 1 and $0!$ is defined as 1. For example, $5!$ is the product $5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$, which is equal to 120.

The factorial of an integer (number in the following example) greater than or equal to zero can be calculated iteratively (nonrecursively) using a for statement, as follows:

```
var factorial = 1;

for ( var counter = number; counter >= 1; --counter )

factorial *= counter;
```

A recursive definition of the factorial function is arrived at by observing the following relationship:

$$n! = n \cdot (n - 1)!$$

For example, 5! is clearly equal to 5 * 4!, as is shown by the following equations:

$$5! = 5 \cdot 4 \cdot 3 \cdot 2 \cdot 1$$

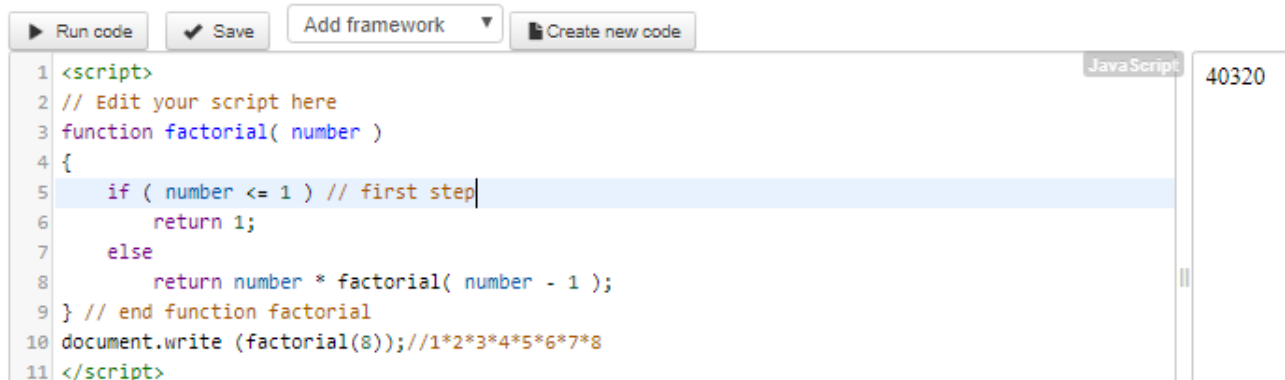
$$5! = 5 \cdot (4 \cdot 3 \cdot 2 \cdot 1)$$

$$5! = 5 \cdot (4!)$$

The recursive function factorial first tests whether a terminating condition is true, i.e., whether number is less than or equal to 1. If so, factorial returns 1, no further recursion is necessary and the function returns. If number is greater than 1, expresses the problem as the product of number and the value returned by a recursive call to factorial evaluating the factorial of number - 1. Note that factorial(number - 1) is a simpler problem than the original calculation, factorial(number).

Example Recursion

1. Go to <https://js.do/>
2. Type and run the code:



The screenshot shows a web-based JavaScript editor interface. At the top, there are buttons for 'Run code', 'Save', 'Add framework', and 'Create new code'. The code editor contains the following JavaScript code:

```
1 <script>
2 // Edit your script here
3 function factorial( number )
4 {
5     if ( number <= 1 ) // first step
6         return 1;
7     else
8         return number * factorial( number - 1 );
9 } // end function factorial
10 document.write (factorial(8)); //1*2*3*4*5*6*7*8
11 </script>
```

On the right side of the editor, the output '40320' is displayed. The code is written in JavaScript, as indicated by the 'JavaScript' label in the top right corner of the editor area.

Uses recursion to calculate and print the factorials of 8

14.2 RECURSION VS. ITERATION



Concepts

Both iteration and recursion are based on a control statement: Iteration uses a repetition statement (e.g., for, while or do...while); recursion uses a selection statement (e.g., if, if...else or switch).

Both iteration and recursion involve repetition: Iteration explicitly uses a repetition statement; recursion achieves repetition through repeated function calls. Iteration and recursion each involve a termination test: Iteration terminates when the loop-continuation condition fails; recursion terminates when a base case is recognized.

Iteration both with counter-controlled repetition and with recursion gradually approaches termination: Iteration keeps modifying a counter until the counter assumes a value that makes the loop-continuation condition fail; recursion keeps producing simpler versions of the original problem until the base case is reached. Both iteration and recursion can occur infinitely: An infinite loop occurs with iteration if the loop-continuation test never becomes false; infinite recursion occurs if the recursion step does not reduce the problem each time via a sequence that converges on the base case or if the base case is incorrect.

14.3 REVIEW EXERCISE

1. _____ terminates when the loop-continuation condition fails:
 - a. Recursion...
 - b. Iteration...
 - c. Loop...
 - d. Function....

2. A function that calls itself:
 - a. Is an incorrect sequence error in JavaScript.
 - b. Could be an example of recursion.
 - c. Will always cause an infinite loop.
 - d. Is called a distortion.

3. An infinite loop is:
 - a. An example of recursion.
 - b. An example of distortion.
 - c. A loop that runs forever.
 - d. A recursive function that draws circles.

LESSON 15 – TESTING AND MODIFICATION

After completing this lesson, you should be able to:

- Understand types of errors in a program like: syntax, logic
- Understand the benefits of testing and debugging a program to resolve errors
- Identify and fix a syntax error in a program like: incorrect spelling, missing punctuation
- Identify and fix a logic error in a program like: incorrect Boolean expression, incorrect data type
- Check your program against the requirements of the initial description
- Identify enhancements, improvements to the program that may meet additional, related needs
- Describe the completed program, communicating purpose and value

15.1 TYPES OF ERRORS



Concepts

A flowchart or a program may have errors. Errors which make the algorithm or program work incorrectly are called **bugs**. For example:

- If one of the decision boxes in a flowchart was labelled incorrectly, with the 'Yes' and 'No' in the wrong places, the algorithm would have an incorrect output. We call this type of error a **bug**.
- If in the magic trick program we divided by 15 rather than 13, we wouldn't get the intended answer at the output. The step dividing by 15 would be called a **bug** in the program.

Serious bugs cause programs to stop working, or **crash**.

A **crash** is when the program stops working completely. Users can lose their data as the result of a crash.

Three types of error commonly occur when programming:

A **syntax error** occurs when a construct in the programming language is incorrectly written. It is an error in the code due to incorrect use of the language such as spelling errors, incorrect punctuation, incorrect naming and referencing. For example, (A+3 is a syntax error because it is missing a closing bracket.

A **logic error** occurs when the program instruction is for an incorrect action. The program appears to be syntactically correct, (i.e. written correctly) but the logic is flawed so when it runs it doesn't do what was expected. In other words, a program may operate correctly but may not do what is required.

A **datatype** error or **typeerror** occurs when you attempt to use an operator or a function on something of the wrong type. For example, if we try to multiply a string and an interger **3*apples** together.

15.2 FINDING ERRORS



Concepts

Syntax Errors

Syntax errors also known as parsing errors are detected immediately by JavaScript when you attempt to run a program. Syntax errors aren't thought of as bugs, since JavaScript will prevent the program from running when it finds a syntax error.

Examples of syntax errors include incorrect syntax, omission of a required colon or bracket, misspelling a keyword, incorrect format for numbers etc. The parser highlights the earliest point in the line where an error is detected

There is a colon at the end of the line which is incorrect. Simply correct this error by removing the colon and re-run the program.

JS.do Online JavaScript Editor

"Edit your code online. Simple, light and fast!"

JavaScript error: Uncaught SyntaxError: Unexpected token ':' on line 3

Code address: <https://js.do/code/>

Description:

Run code Save Add framework Create new code

```
1 <script>
2 // Edit your script here
3 document.write ("Debugging");:
4 </script>
```

JavaScript

Corrected:

Run code Save Add framework Create new code

```
1 <script>
2 // Edit your script here
3 document.write ("Debugging");
4 </script>
```

JavaScript

Debugging

Logic Errors

Logic errors are not always easy to find. Consider the logic in this program to display an instruction to put a cake in an oven when the oven reaches the correct temperature.

```
1 <script>
2 // Edit your script here
3 var correct_temp="180C"
4 var current_temp="75C"
5 if (current_temp!=correct_temp)
6     document.write("put cake in oven");
7 else
8     document.write("wait until the oven is at the correct temperature");
9 </script>
```

```
1 <script>
2 // Edit your script here
3 var correct_temp="180C"
4 var current_temp="75C"
5 if (current_temp!=correct_temp)
6     document.write("put cake in oven");
7 else
8     document.write("wait until the oven is at the correct temperature");
9 </script>
```

JavaScript put cake in oven

The logic is wrong. To correct we need to change the != (Not equal) symbol to the == (Equals) symbol.

```
1 <script>
2 // Edit your script here
3 var correct_temp="180C"
4 var current_temp="75C"
5 if (current_temp==correct_temp)
6     document.write("put cake in oven");
7 else
8     document.write("wait until the oven is at the correct temperature");
9 </script>
```

```
1 <script>
2 // Edit your script here
3 var correct_temp="180C"
4 var current_temp="75C"
5 if (current_temp==correct_temp)
6     document.write("put cake in oven");
7 else
8     document.write("wait until the oven is at the correct temperature");
9 </script>
```

JavaScript wait until the oven is at the correct temperature

Consider the example below and see if you can correct the logic:

```
1 <script>
2 // Edit your script here
3 var Age=prompt("How old is he?")
4 Age=parseInt(Age);
5 if (Age>13 && Age<19)
6     document.write("He's a teenager");
7 else
8     document.write("He's not a teenager");
9 </script>
```

There is a problem with the logic test in this program.

The logic test will work correctly for age inputs 14, 15, 16, 17 and 18 as the program will correctly print “He’s a teenager” for each input.

But for Age inputs 13 and 19 it will print “He’s not a teenager”

To fix the program:

1.

```
<script>
// Edit your script here
var Age=prompt("How old is he?")
Age=parseInt(Age);
if (Age>13 && Age<19)
    document.write("He's a teenager");
else
    document.write("He's not a teenager");
</script>
```

2. Type the code above correctly to fix the bug, by changing the line of code beginning with 'if'.
3. Check your solution by running your edited program.
4. Test values for age such as 12, 13, 14, 18, 19 and 20 to see if you get the expected output.

Example: Find and Fix logic and syntax errors

The following program has two syntax errors and one logic error.

1. Type in the program as shown below.

```
1 <script>
2 // Edit your script here
3 var Age=Prompt("How old are you?");
4 Age=parseInt(Age);
5 if (Age>65 && Age<70)
6     document.write("You're a teenager");
7 else
8     document.write("You're not a teenager");
9 </script>
```

```
<script>
// Edit your script here
var Age=Prompt("How old are you?");
Age=parseInt(Age);
if (Age>65 && Age<70)
    document.write("You're a teenager");
else
    document.write("You're not a teenager");
</script>
```

2. Fix the syntax errors.
3. Fix the logic error too.
4. Run the program to check your work.

15.3 TESTING AND DEBUGGING A PROGRAM



Concepts

Checking for Errors

A programmer might notice that their program has a logic error, or they might not. It is important to detect logic errors. One way to do this systematically is by **testing**.

Testing

Running a computer program using different inputs to find if there are logic errors.

In testing, the tester needs to know what the results should be. They may refer to the requirements in a specification to check that the program is doing what it is supposed to.

After fixing a bug the program will be tested again to make sure the bug really has been fixed.

Testing Against Requirements

Best practice is to test a program against a specification document written at the design stage. The specification gives clarity about what the program should do. It lists requirements for the program.

A specification for a program will usually number the requirements. This is a possible example:

1.3.1: The program SHALL dim the screen if the computer has not been used for one minute.

A corresponding plan for testing would include a step for testing that requirement. The plan would stipulate leaving the program running for one minute and confirming that the screen does go dim after that time. Each requirement should have some corresponding tests.

In this example, if after one minute the program crashes, there is a bug in the program. However, if nothing special happens after a minute, one could argue that there is no bug.

This shows how systematic testing against requirements is valuable. Testing would have identified a requirement that had not been implemented. Testing against requirements also helps in finding problems that only show when certain parts of the program are used. Testing in this way may find problems that would not otherwise have been found.

Finding Bugs

When a program gives the wrong or unexpected results, or if the program crashes unexpectedly with an error, it is not always obvious why that has happened.

When a program gives the wrong answer, it may not always be enough to just look at the code to find the bug.

One way to find out more about the bug is to include some extra diagnostics in the code. These could be additional print statements in the program to display the values of variables at various points in the program. These extra print statements show how the values were changing.

The work of finding out the cause of a bug is called **debugging**.

Debugging

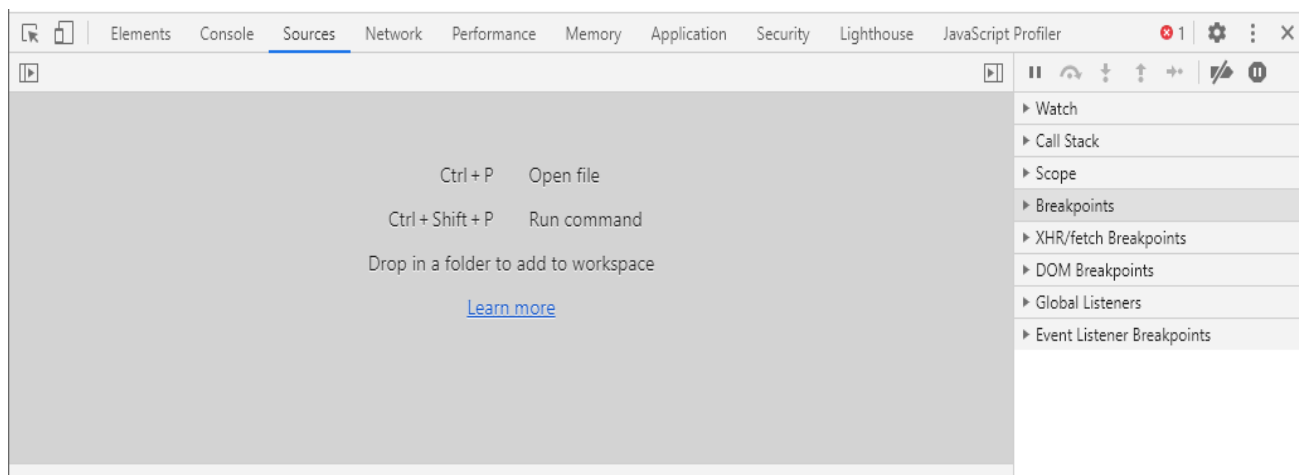
Running a faulty program with extra diagnostic information to work out where the mistakes in the code are, and then fix those mistakes.

One other way to diagnose a program is to run it 'under a debugger'. This is a tool that can be used to run the program one line at a time. .

Debugging may uncover the bug and also show when the bug first occurs. Without debugging, it may not be clear where the problem code is.

Debugging in Chrome

- Open your page in Chrome.
- Turn on developer tools with F12 (Mac: Cmd+Opt+I).
- Select the Sources pane

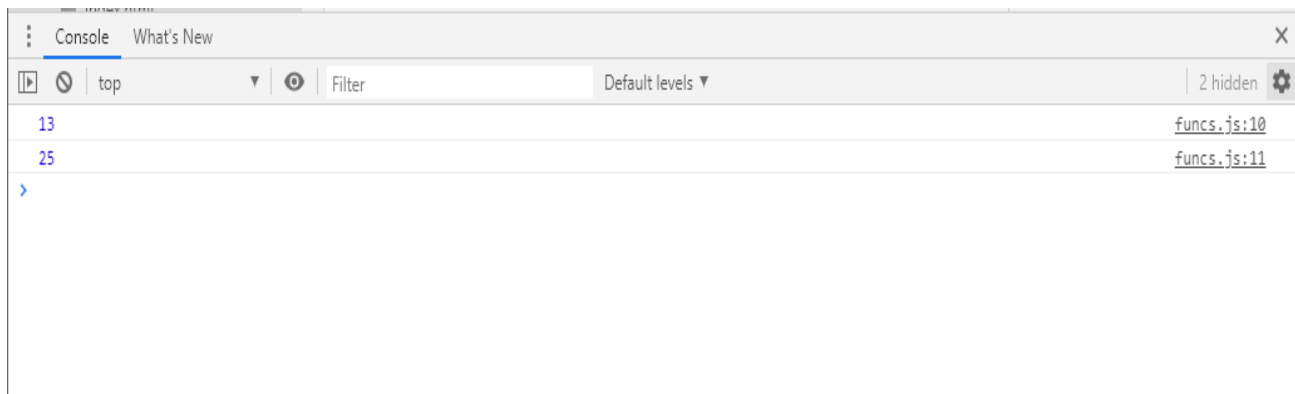


- The above button opens the tab with files.
- Click it and select your js in the tree view.



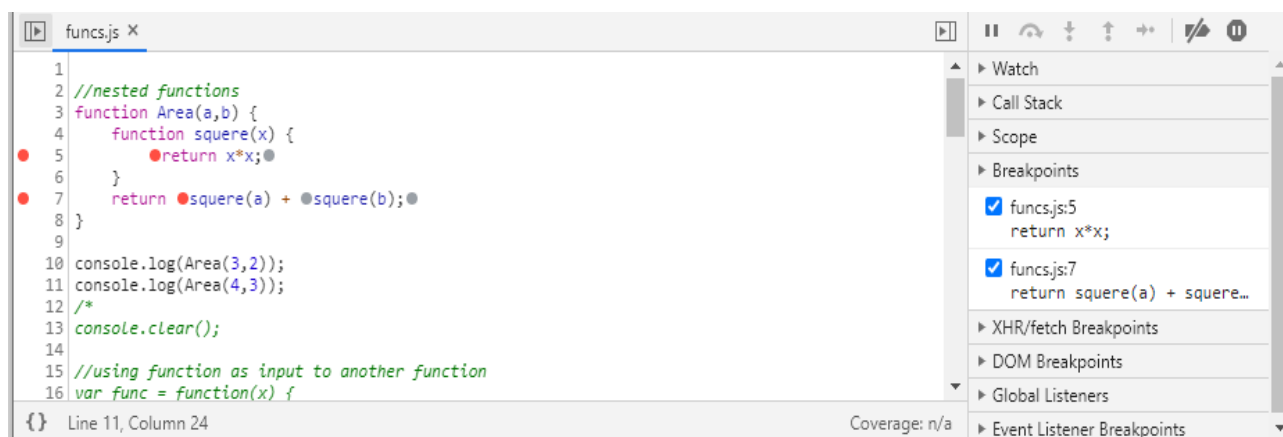
- The Sources panel has 3 parts:
 - The **File Navigator** pane lists HTML, JavaScript, CSS and other files, including images that are attached to the page. Chrome extensions may appear here too.
 - The **Code Editor** pane shows the source code.
 - The **JavaScript Debugging** pane is for debugging, we'll explore it soon.

- If we press Esc, then a console opens below. We can type commands there and press Enter to execute.
- After a statement is executed, its result is shown below



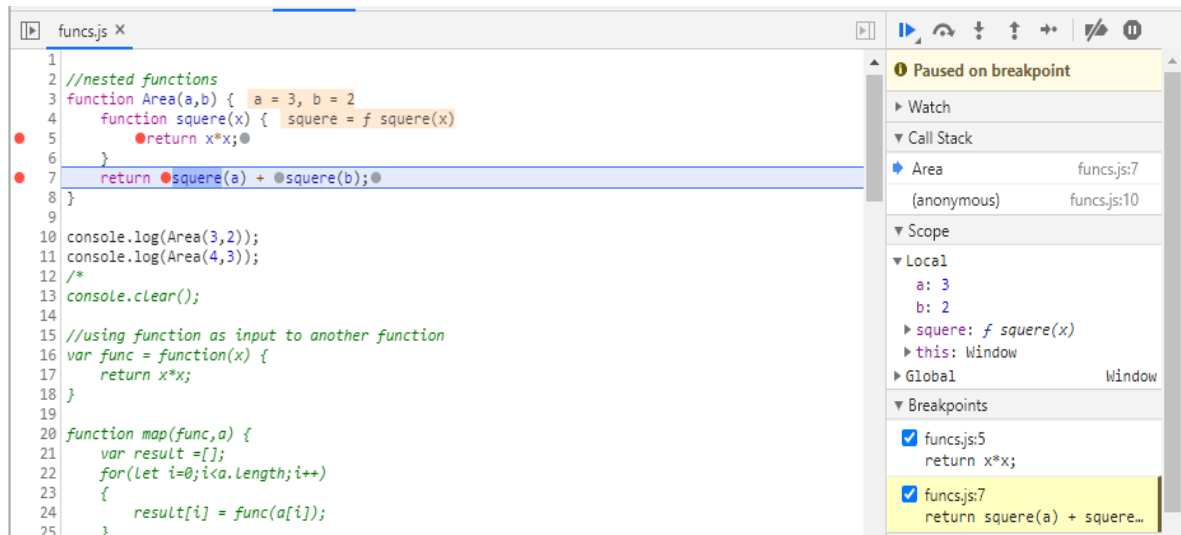
- **Breakpoints**


- Let's examine what's going on within the code of the example page. In js, click at line number 4.
- Also click on the number for line 7.
- It should look like this:



- A breakpoint is a point of code where the debugger will automatically pause the JavaScript execution.
- While the code is paused, we can examine current variables, execute commands in the console etc. In other words, we can debug it.
- We can always find a list of breakpoints in the right panel. That's useful when we have many breakpoints in various files. It allows us to:
 - Quickly jump to the breakpoint in the code (by clicking on it in the right panel).

- Temporarily disable the breakpoint by unchecking it.
- Remove the breakpoint by right-clicking and selecting Remove.
- ...And so on.
- Press F5 (Windows, Linux) or Cmd+R (Mac).
- As the breakpoint is set, the execution pauses at the 5th line:



- **Watch** – shows current values for any expressions.
 - You can click the plus + and input an expression. The debugger will show its value at any moment, automatically recalculating it in the process of execution.
- **Call Stack** – shows the nested calls chain.
 - At the current moment the debugger is inside `hello()` call, called by a script in `index.html` (no function there, so it's called "anonymous").
 - If you click on a stack item (e.g. "anonymous"), the debugger jumps to the corresponding code, and all its variables can be examined as well.
- **Scope** – current variables.
 - Local shows local function variables. You can also see their values highlighted right over the source.
 - Global has global variables (out of any functions).
 - There's also this keyword there that we didn't study yet, but we'll do that soon.
-  **Resumes the execution.** If there are no additional breakpoints, then the execution just continues and the debugger loses control.

- The execution has resumed, reached another breakpoint inside js and paused there. Take a look at the “Call Stack” at the right. It has increased by one more call.
- Step: run the next command, hotkey F9
- Step over: run the next command, but don’t go into a function, hotkey F10.
- Step into, hotkey F11.
- Step out: continue the execution till the end of the current function, hotkey Shift+F11.

15.4 IMPROVING A PROGRAM

Concepts

Once a program is written and is successful, there is often pressure to create a new and improved version of it.

Commercial considerations usually mean that a complete rewrite of the program is not a good idea. A better plan is usually to make small incremental improvements that make the program more useful to more people. Such changes help the code to solve a more general problem than the initial version.

Suitability for Different Countries

Recall that in the lesson on computational thinking we looked at generalisation. We saw that a washing machine design might be generalised by making it suitable for more countries. This is very common in software too.

Software may benefit from translation of the strings into other languages. Usually programmers attempt to use the same source code, but make the code more flexible so that it can use different strings for different languages as indicated by a parameter.

Generalising code to make it suitable for different countries is sometimes called **localisation**. Localisation can go deeper than simply translation. A recipe program might support different kinds of measurement. For example, a recipe program might support temperature measured in degrees Fahrenheit or in degrees Celsius. This would help it to be useful in more countries.

Work on localisation may also draw attention to other related opportunities for generalisation.

Localisation work on providing quantities of ingredients in different measurement units, as needed in different countries, could lead on to other useful generalisations. An improved program could calculate quantities in different units. It could allow the user of the program to request adjustments to portion size and number of portions, with the program recalculating the quantities.

Solving a Larger Problem

In the lesson on computational thinking recall that we saw that a solution to the problem of running a musical festival could be generalised. Experience in running a music festival could contribute to solving the more ambitious problem of running a musical tour.

In looking for opportunities to improve a recipe program, small additions that, for example, count calories, or add up nutritional information, could generalise the recipe program and make it part of a larger computer program for health and nutrition.

Successful improvement to a program needs to balance ambition with practicality. A large change in functionality of a program may require a rewrite, or may require a new library to bring in new functionality. Large changes, such as adding voice recognition into a program go beyond what can be achieved with small incremental improvements.

Presenting a Program

A programmer's view of a program is very different to the view of the program a user or a manager needs to have. When presenting a program, be aware of the audience and what aspects of the program are relevant to them. This is particularly important when communicating proposed improvements. Pseudocode and flowcharts may be relevant in showing how an algorithm works. They do little to communicate purpose and value.

Communicating Purpose: When communicating the purpose of a program, explain what problem the program solves. Generally programs are useful in parts of a problem that are repeated again and again. Computers are good at automation.

In communicating purpose, it may be useful to also communicate what parts of a problem a program does *not* solve.

A word processing program saves an author from typing the same text out many times as they work through different versions of a document. A word processing program does little to help an author find the right words and find ideas for characters and plot.

Communicating Value: The value of a program or improvements to a program depend on who will use it in the market and on competition. Managers may want to know if a similar program exists already, or if the new program can easily be copied by others.

If the algorithms are new, creative, innovative, it may be possible to patent them and so protect them from being copied.

If the program uses a proprietary file format, i.e. owned by the programmer of his/her company, that stores data in a way that other programs can't easily read, that may inhibit other companies from making their own version.

In communicating the value of a program, highlight what is unique about it. Show that the unique aspects are of value to end users. A valuable program is one that saves the end user time and money. Where possible support your claims about the value of a program with numbers.

15.5 REVIEW EXERCISE

1. Which expression has a syntax error?
 - a. $2+4+8$
 - b. $(2+4)*8$
 - c. $2/4/8$
 - d. $2+(4*8))$

2. A logic test was intended to test if variable c, d and e are in order with c smallest and e largest. Which of the following has a logic error:
 - a. if (c<d) && (d<e) :
 - b. if (e>d) && (d>c) :
 - c. if (c#d) && (d#e) :
 - d. if (e<d) && (d<c) :

3. Program testing is:
 - a. Checking with the customer that the specification is what they want.
 - b. Fixing a bug when a user reports an error.
 - c. When someone who has more experience has a look at the source code for a program and helps improve it.
 - d. A formal approach to finding bugs in a program, by running the program with different inputs.

4. Debugging is:
 - a. Checking a program against its requirements to see if it does what it is supposed to do.
 - b. Enhancing a program by adding new features.
 - c. Work done to track down the cause of a problem and fix it.
 - d. Enhancing a program by taking out features which aren't needed anymore.

5. A software company wants to make incremental improvements to version 1 of its program for managing recipes. Which of the following is a good way to identify possible enhancements?
 - a. Add code for drawing a fern recursively.
 - b. Use Google to search for "robots AND cooking"
 - c. Look for small changes that would make the program useful to more people.

6. When describing the commercial value of a new program, the best way is:
 - a. Show as detailed a flowchart as possible to show how complex the program is.
 - b. Describe the benefit to users and back it up with numerical data if you can.
 - c. Explain that the program is new, and therefore it must be of value.
 - d. Explain that it uses only well known, widely used, reliable algorithms.

ICDL Syllabus

Ref.	Task Item	Location	Ref.	Task Item	Location
1.1.1	Define the term computing.	1.1 <i>Computational Thinking</i>	2.1.4	Use abstraction to filter out unnecessary details when analysing a problem.	1.1 <i>Computational Thinking</i>
1.1.2	Define the term computational thinking.	1.1 <i>Computational Thinking</i>	2.1.5	Understand how algorithms are used in computational thinking.	1.1 <i>Computational Thinking</i> 1.2 <i>Instructing a Computer</i>
1.1.3	Define the term program.	1.2 <i>Instructing a Computer</i>	2.2.1	Define the programming construct term sequence. Outline the purpose of sequencing when designing algorithms.	3.1 <i>Steps in an Algorithm</i>
1.1.4	Define the term code. Distinguish between source code, machine code.	2.2 <i>Computer Languages</i>	2.2.2	Recognise possible methods for problem representation like: flowcharts, pseudocode.	3.1 <i>Steps in an Algorithm</i> 3.2 <i>Flowcharts</i>
1.1.5	Understand the terms program description and specification.	2.3 <i>Text About Code</i>	2.2.3	Recognise flowchart symbols like: start/stop, process, decision, input/output, connector, arrow.	3.2 <i>Flowcharts</i>
1.1.6	Recognise typical activities in the creation of a program: analysis, design, programming, testing, enhancement.	2.4 <i>Stages in Developing a Program</i>	2.2.4	Outline the sequence of operations represented by a flowchart, pseudocode.	3.2 <i>Flowcharts</i> 3.3 <i>Pseudocode</i>
1.1.7	Understand the difference between a formal language and a natural language.	2.1 <i>Precision of Language</i>	2.2.5	Write an accurate algorithm based on a description using a technique like: flowchart, pseudocode.	3.2 <i>Flowcharts</i> 3.3 <i>Pseudocode</i>
2.1.1	Outline the typical methods used in computational thinking: decomposition, pattern recognition, abstraction, algorithms.	1.1 <i>Computational Thinking</i>	2.2.6	Fix errors in an algorithm like: missing program element, incorrect sequence, incorrect decision outcome.	3.4 <i>Fixing Algorithms</i>
2.1.2	Use problem decomposition to break down data, processes, or a complex problem into smaller parts.	1.1 <i>Computational Thinking</i>			
2.1.3	Identify patterns among small, decomposed problems.	1.1 <i>Computational Thinking</i>			

Ref.	Task Item	Location	Ref.	Task Item	Location
3.1.1	Describe the characteristics of well-structured and documented code like: indentation, appropriate comments, descriptive naming.	9.1 <i>Readable Code</i> 9.2 <i>Comments</i> 9.3 <i>Organisation of Code</i> 9.4 <i>Descriptive Names</i>	3.2.4	Use appropriately named variables in a program for calculations, storing values.	6.2 <i>Variables</i> 9.4 <i>Descriptive Names</i>
3.1.2	Use simple arithmetic operators to perform calculations in a program: +, -, /, *.	5.1 <i>Performing Calculations with JavaScript</i>	3.2.5	Use data types in a program: string, character, integer, float, Boolean.	6.1 <i>Data Types</i> 6.2 <i>Variables</i> 6.3 <i>Beyond Numbers</i> 7.4 <i>Booleans and Variables</i>
3.1.3	Understand the precedence of operators and the order of evaluation in complex expressions. Understand how to use parenthesis to structure complex expressions.	5.2 <i>Precedence of Operators</i> 7.5 <i>Putting it all together</i>	3.2.6	Use an aggregate data type in a program like: array, list, tuple.	8.1 <i>Aggregate Data Types</i> 8.2 <i>Lists</i> 8.3 <i>Tuples</i>
3.1.4	Understand the term parameter. Outline the purpose of parameters in a program.	11.3 <i>Procedures and Functions</i>	3.2.7	Use data input from a user in a program.	6.3 <i>Beyond Numbers</i> 10.3 <i>IF ELSE Statement</i>
3.1.5	Define the programming construct term comment. Outline the purpose of a comment in a program.	9.2 <i>Comments</i>	3.2.8	Use data output to a screen in a program.	4.2 <i>Exploring JavaScript</i> 6.2 <i>Variables</i>
3.1.6	Use comments in a program.	9.2 <i>Comments</i>	4.1.1	Define the programming construct term logic test. Outline the purpose of a logic test in a program.	10.2 <i>IF Statement</i>
3.2.1	Define the programming construct term variable. Outline the purpose of a variable in a program.	6.2 <i>Variables</i>	4.1.2	Recognise types of Boolean logic expressions to generate a true or false value like: =, >, <, >=, <=, <>, !=, ==, AND, OR, NOT.	7.2 <i>Comparison Operators</i> 7.3 <i>Boolean Operators</i> 7.5 <i>Putting it all together</i>
3.2.2	Define and initialise a variable.	6.2 <i>Variables</i>	4.1.3	Use Boolean logic expressions in a program.	10.2 <i>IF Statement</i> 10.3 <i>IF ELSE Statement</i>
3.2.3	Assign a value to a variable.	6.2 <i>Variables</i>	4.2.1	Define the programming construct term loop. Outline the purpose and benefit of looping in a program.	12.1 <i>Looping</i>

Ref.	Task Item	Location	Ref.	Task Item	Location
4.2.2	Recognise types of loops used for iteration: for, while, repeat.	12.1 Looping 12.2 Looping with Variables 12.3 Variations on Loops	4.5.1	Understand the term event. Outline the purpose of an event in a program.	13.2 Events
4.2.3	Use iteration (looping) in a program like: for, while, repeat.	12.1 Looping 12.2 Looping with Variables 12.3 Variations on Loops 12.4 Putting it all together	4.5.2	Use event handlers like: mouse click, keyboard input, button click, timer.	13.4 Pygame Library 13.6 Drawing Using the Libraries
4.2.4	Understand the term infinite loop.	12.3 Variations on Loops	4.5.3	Use available generic libraries like: math, random, time.	13.2 Standard Libraries
4.2.5	Understand the term recursion.	14.1 Recursion	5.1.1	Understand the benefits of testing and debugging a program to resolve errors.	15.3 Testing and Debugging a Program
4.3.1	Define the programming construct term conditional statement. Outline the purpose of conditional statements in a program.	10.2 IF Statement	5.1.2	Understand types of errors in a program like: syntax, logic.	15.1 Types of Errors
4.3.2	Use IF...THEN...ELSE conditional statements in a program.	10.2 IF ELSE Statement	5.1.3	Run a program.	4.3 Save a Program
4.4.1	Understand the term procedure. Outline the purpose of a procedure in a program.	11.3 Functions and Procedures	5.1.4	Identify and fix a syntax error in a program like: incorrect spelling, missing punctuation.	15.2 Finding Errors 15.3 Testing and Debugging a Program
4.4.2	Write and name a procedure in a program.	11.3 Functions and Procedures 12.4 Putting it all together	5.1.5	Identify and fix a logic error in a program like: incorrect Boolean expression, incorrect data type.	15.2 Finding Errors 15.3 Testing and Debugging a Program
4.4.3	Understand the term function. Outline the purpose of a function in a program.	11.3 Functions and Procedures	5.2.1	Check your program against the requirements of the initial description.	2.4 Stages in Developing a Program 15.3 Testing and Debugging a Program
4.4.4	Write and name a function in a program.	11.3 Functions and Procedures 12.4 Putting it all together	5.2.2	Describe the completed program, communicating purpose and value.	15.4 Improving a Program
			5.2.3	Identify enhancements, improvements to the program that may meet	15.4 Improving a Program

Ref.	Task Item	Location
	additional, related needs.	

Congratulations! You have reached the end of the ICDL Computing book. You have learned about the skills and competences relating to computational thinking and coding that will guide you through the process of problem solving and creating simple computer programs:

- Understand key concepts relating to computing and the typical activities involved in creating a program.
- Understand and use computational thinking techniques like problem decomposition, pattern recognition, abstraction and algorithms to analyse a problem and develop solutions.
- Write, test and modify algorithms for a program using flowcharts and pseudocode.
- Understand key principles and terms associated with coding and the importance of well-structured and documented code.
- Understand and use programming constructs like variables, data types, and logic in a program.
- Improve efficiency and functionality by using iteration, conditional statements, procedures and functions, as well as events and commands in a program.
- Test and debug a program and ensure it meets requirements before release.

Having reached this stage of your learning, you should now be ready to undertake an ICDL certification test. For further information on taking this test, please contact your ICDL test centre.

PeopleCert Education

Panepistimiou 39

10564 Athens

Greece

For Coding Bootcamp Students | No distribution allowed.

