DIXONS SIXTH FORM ACADEMY

SUMMER WORK:

A LEVEL COMPUTER SCIENCE

2023

Contents

About the Summer Work

This booklet contains a number of tasks that students are expected to complete to a good standard in order to be able to be enrolled in this subject.

The tasks are to complete some Programming.

Please do the following:

1. complete the programs as directed
2. copy them onto a Word Document
3. send them as an MS Word document to jpatel@dixons6a.com.

The work handed in should have your name on it and all tasks should have a heading to indicate the task number – e.g. Task 1, Task 2, etc

This booklet also contains significant additional information and a range of CHALLENGE tasks. We would encourage you to complete all the tasks, but you do not need to bring any CHALLENGE tasks to your enrolment interview.

Video help files for this can be found on this YouTube playlist.

https://www.youtube.com/playlist?list=PL4XMAC0UmRoI9l7JMSipGrUOY2eH6EX65

There is also a section on additional work for learning some web programming. This is not compulsory but it would be very good if you could complete this!

What you will need

I would recommend that you install PyCharm Educational version onto your computer. You can get this from the following link: https://www.jetbrains.com/edu-products/download/#section=pycharm-edu

Alternatively, you will need to install PYTHON 3 on your computer. You can get this from https://www.python.org/downloads/

Any problems then please email jpatel@dixons6a.com

Welcome to Computer Science

Subject outline

One perception of Computer Science is that it is forever changing. Technology has evolved so much that we live in a world where it is difficult to image life without computerised systems. However, this evolving technology is built upon some solid foundations that have stood since its inception – this is Computer Science.
There are three main strands to Computer Science, and these form our "Big Picture Questions":

- How do computers work?
- How do computers communicate and work together?
- How do we make computers work for us?

Everything that we will do falls under one of these questions.
At Dixons we study AQA A level Computer Science which is breaks down these themes further and examines them in three parts:

1. Paper 1:

   This paper tests your ability to program, as well as some theoretical knowledge of Computer Science around the question "**How do we make computers work for us?**"
2. Paper 2:

   This paper tests theoretical knowledge of Computer Science around the questions "**How do computers work?**" and "**How do computers communicate and work together?**"
3. NEA:

   The non-exam assessment assesses your ability to use the knowledge and skills gained through the course to solve or investigate a practical problem. You will be expected to follow a systematic approach to problem solving,

What an Excellent Student "Looks Like"

To study computer science and get the best out of it you need to have three main attributes:

1. You need to have, or be developing, excellent coding skills.

   These are:
   - good programming habits,
   - good use of programming techniques
   - being able to problem solve quickly
   - being able to code across several programming languages

2. You need to have excellent mathematical ability.

   In reality, computer science is applied mathematics so you need to have a very good grasp of it and be agile in your thinking!

3. You need to be hard working, concentrate well and be resilient

   Computer Scientists are design engineers and often have to work at a problem over a long period of time. Sometimes things can go wrong and you have to start again. You will get frustrated and you need to show good resilience.
   A lot of work and practice needs to take place outside of lesson time so you need to be able to concentrate well wherever you are.

Careers & Higher Education

Need some Inspiration?

Computer Science matters!!! https://www.youtube.com/watch?v=_IiPLtP-jm8&list=PLzdnOPI1iJNfpD8i4Sx7U0y2MccnrNZuP

Computer Science is everywhere!!
https://www.youtube.com/watch?v=QvyTEx1wyOY&feature=youtu.be

Computer Science is for everyone!!! https://www.youtube.com/watch?v=mFPg96gdPkc

Many careers are linked to Computer Science. Many students who take this course want to go on to a career as a programmer – maybe you could be the next Bill Gates or Mark Zuckerberg!

In terms of the local area, there are many new niche sectors beginning to develop that will require computer scientists in new media and telecoms, research and development in heavy and light industries and growing Small/Medium sized Enterprises.

Below are some link to places that you can look for more information and inspiration:

A sample of Universities

Computer Science at Oxford University

https://www.youtube.com/watch?v=JG-mHQX8eZw

Computer Science at Cambridge University

https://www.youtube.com/watch?v=94NgNNs1O3Q&feature=youtu.be

Computer Science at Manchester University

https://www.manchester.ac.uk/study/undergraduate/courses/2023/00560/bsc-computer-science/course-details/#course-profile

Some Different Careers

What can you do with a Computer Science Degree?

https://www.prospects.ac.uk/careers-advice/what-can-i-do-with-my-degree/computer-science

Links to key information:

Click here for the AQA Specification for A Level Computer Science: https://www.aqa.org.uk/subjects/computer-science-and-it/as-and-a-level/computer-science-7516-7517/subject-content-a-level

Summer work tasks

Task 1: Different Programming Languages

There are many programming languages that are used in technology.

This task is to find out a little bit different languages - watch this video:

https://youtu.be/7bE2mI4ePeU

Summarise the contents of the video in a table with the following headings – one example is already completed for you to follow.

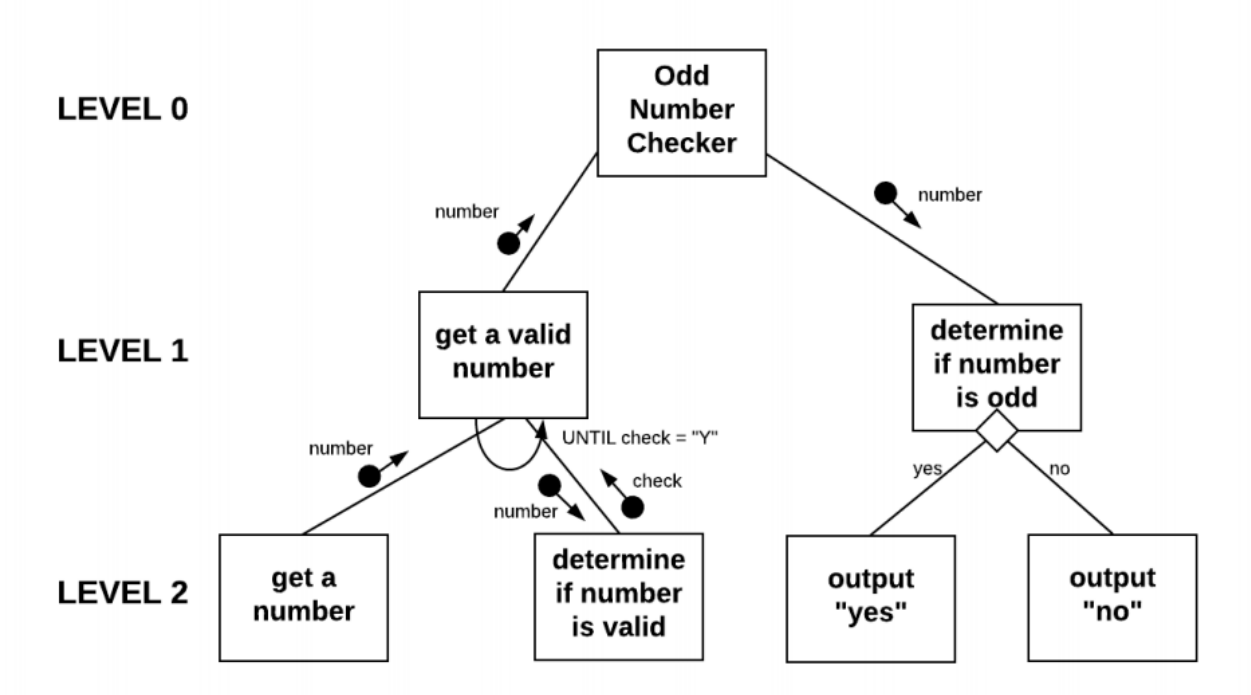
Language	**Where this language is used**
JavaScript	This language is used on webpages to control what happens when users interact with it

Structured Programming

The initial approach to programming that we use at Dixons Sixth Form Academy is called Structured Programming. This means that our programs will always be broken down into

1. a main routine to "control" the sequence
2. and then subroutines (functions) to be "workers" that complete tasks for us

This diagram shows this for a simple problem for checking odd numbers:



The main "control" routine is on Level 0 and it uses "worker" subroutines to complete tasks.

So, from the diagram you can see that the main routine calls workers to:

- get a valid number
- determine if the number is odd

Let's try writing and running a program to see this in action.

Task 2: Starting to Program in PYTHON

Program: Odd number checker

Using PYHTON (installed on your computer) read through the explanations, type up the program and run it.

You must complete Steps 1 to 4 BEFORE trying to run the program.

Step 1: Write the MAIN Routine

```
def main():
    # MAIN ROUTINE
    #  LOCAL VARIABLES
    #    number  string    to hold the inputted number

    # get the number
    number = getNatural("Please enter a number greater than 0: ")

    # determine if the number is odd
    if determineIfOdd(number):
        print("The number is odd")
    else:
        print("The number is even")
    #end if
# END
```

Things to note:

1. The name of the routine is preceded by a `def` key word. This is a " `def`inition for a subroutine". This is the control routine and is ALWAYS called `main`.
2. There are comments in the code – this is GOOD PRACTICE and we want all of your programs to be written with these comments in them so:
    a. All variables used should be listed with the type that they will be and what data they will hold. For example, `number` is a `string` (text) and will `hold the inputted number`.
    b. Sections of code should be commented. This will aid debugging (finding errors) and will help you to direct the flow the program.
    c. PYTHON doesn't use and END statements for IFs or LOOPS but these are included here as comments. This will help you to read the code and understand it better.

Step 2: write the subroutine getNatural()

```
def getNatural(message):
    # SUBROUTINE  RETURNS integer
    #  LOCAL VARIABLES
    #    valid   boolean   to indicate state of input
    #    number  string    to hold the inputted number

    # initialise switch for validity
    valid = False
    while not valid:
        # get number
        number = input(message)
        # determine if the number is an integer
        if not number.isdigit():
            displayMessage("invalid number, try again")
        else:
            valid = True
        # END IF
    # END WHILE

    return int(number)
# END
```

Things to note:

1. Notice that the first comment identifies the section of code as a `subroutine` and states that it will `return an integer`.
2. All variables are listed
3. Sections of code are commented.
4. END IF and END WHILE are written in as comments.
5. The number is tested for being a natural number by using a string method called `isdigit().` This method will return `true` if the string contains just digits (0-9) and `false` if it contains something else. **This give us just Natural numbers (greater than 0) not integers**.
6. The number is returned as an INTEGER to the calling routine by using the built-in `int()` function. This takes a `string` as a `parameter` and `returns` an `integer`.

Step 3: Write the subroutine determineIfOdd()

```
def determineIfOdd(number):
    # SUBROUTINE  RETURNS odd : boolean
    #  PARAMETERS
    #    number  integer   holds the number to test
    #  LOCAL VARIABLES
    #    odd   boolean     to indicate state of test

    # determine if odd
    odd = False
    if number % 2 != 0:
        odd = True
    # end if

    return odd
#end def
```

Things to note:

1. Notice that the first comment identifies the section of code as a `subroutine` and states that it will `return a value`.
2. This subroutine has a `parameter` that is written `in the brackets`.
3. All `parameters` and `variables` are listed
4. Sections of code are commented.
5. END IF is written in as a comment.

Step 4: enable the program to run

```
58
59  # run the main program
60  main()
61
```

Things to note:

1. This line of code runs the main routine. Without it, the program will not run!

Step 5: Run your code and Test it

Testing the Program

It is always important to test the program that you write to ensure that it works. This is done using a Test Plan:

Number	Type of Test	Expected Result	Success
53	Real	YES	
10	Real	NO	
2122345	Real	YES	
X	Erroneous	You will be asked to enter the number again	

Step 6: In Word

1. Copy the entire code to a Word Document under the heading Task 2.
2. Draw a Test Plan table and put suitable tests into it
3. Carry out the tests and indicate on the table if they were successful or not.

Task 3: Determine if a number is a multiple of 7

Your task is to adapt the program that you have just written so that instead of determining if a number is odd, it determines if it is a multiple of 7.

Repeat the same steps as for Task 2 but write a new subroutine called `determineIfMultipleOfSeven()` that will perform the task required.

Final Step: In Word

1. Copy the entire code to a Word Document under the heading Task 3.
2. Draw a Test Plan table and put suitable tests into it
3. Carry out the tests and indicate on the table if they were successful or not.

Task 4: Determine if a year is a century

Your task is to adapt the program that you have just written so that instead of determining if a number is a multiple of seven, it determines if the year entered is a century.

Repeat the same steps as for Task 2 and 3 but write a new subroutine called `determineIfCentury()` that will perform the task required.

Final Step: In Word:

1. Copy the entire code to a Word Document under the heading Task 3.
2. Draw a Test Plan table and put suitable tests into it
3. Carry out the tests and indicate on the table if they were successful or not.

Task 5: Determine if a year is Leap Year

CHALLENGE YOURSELF

This is a harder task and requires a little more thought.

A year is a leap year if it is divisible by 4. However, if the year is a century, it is only a leap year if it is divisible by 400.

Your task is to adapt the program that you have just written so that instead of determining if a year is a century, it determines if the year entered is a leap year.

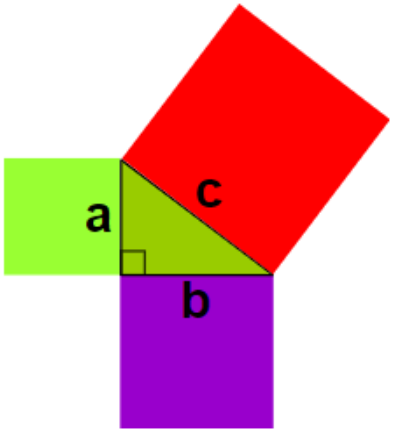
Repeat the same steps as for Task 2, 3 & 4 but write a new subroutine called `determineIfLeapYear()` that will perform the task required.

Final Step: In Word:

1. Copy the entire code to a Word Document under the heading Task 5.
2. Draw a Test Plan table and put suitable tests into it
3. Carry out the tests and indicate on the table if they were successful or not.

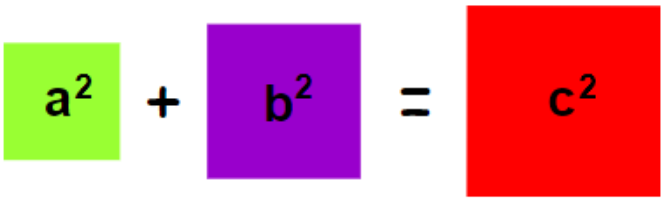
Task 6: Calculate the Hypotenuse for a right-angled triangle

From GCSE Mathematics you will understand how Pythagoras' Theorem works:



It is called "Pythagoras' Theorem" and can be written in one short equation:

$$a^2 + b^2 = c^2$$

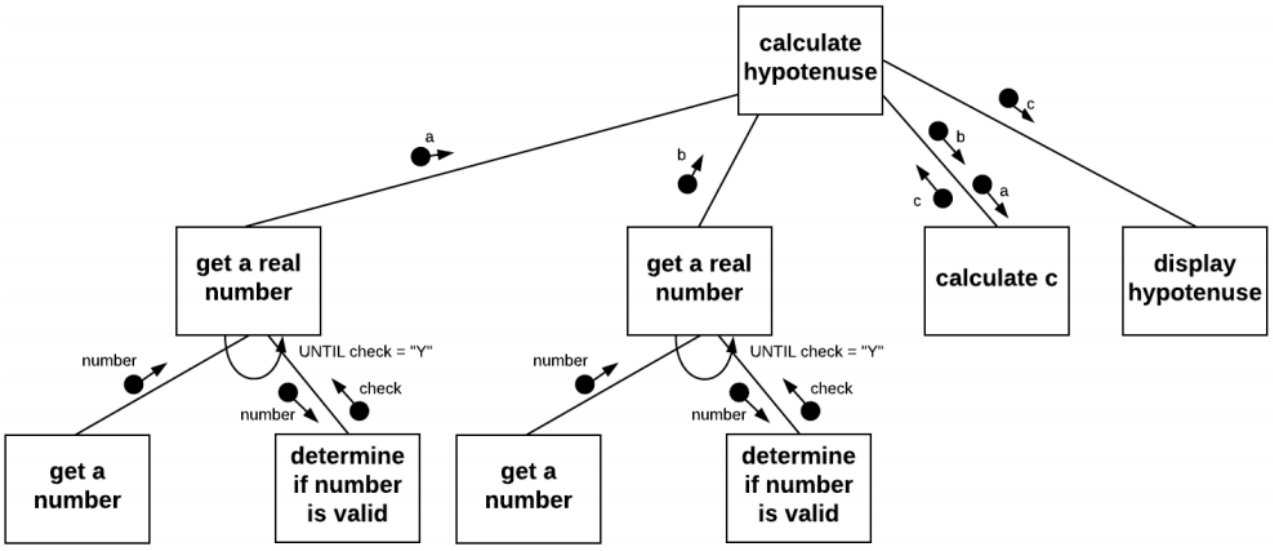


Note:

- **c** is the **longest side** of the triangle
- **a** and **b** are the other two sides

Here both `a` and `b` need validating as real (decimal) numbers that are larger than 0.

Rather than clutter up the main algorithm with the validation we can use the same SUBROUTINE TWICE.



Step 1: write the MAIN routine

You should be able to do this yourself but here is the code:

```
import math
def main():
    # MAIN ROUTINE
    #  LOCAL VARIABLES
    #    a    real  to hold the length of side a
    #    b    real  to hold the length of side b
    #    c    real  to hold the length of the hypotenuse

    # get a and b
    a = getPositiveReal("Please enter a value for a: ")
    b = getPositiveReal("Please enter a value for b: ")

    # calculate the hypotenuse
    c = calculateHypotenuse(a, b)

    # display the hyptoenuse
    displayReal("The hypotenuse is: ", c, 3)

# END
```

Step 2: write the subroutine testForReal()

This subroutine contains a new construct to help trap errors so that they don't crash the program. This is called TRY and EXCEPT.

Whatever is in the TRY block will be tried – if there is an error, the EXCEPT block takes over. So the code works like this by testing a conversion of the number to a float (real number):

```
def testForReal(number):
    # SUBROUTINE   RETURNS boolean
    #  PARAMETERS
    #    number   string  holds the number string being tested
    #  LOCAL VARIABLES
    #    real     real    holds the converted real

    # set switch
    valid = True
    try:
        # try converting the number to a real
        real = float(number)
    except:
        # set valid to false since there has been an error
        valid = False
    # END TRY

    # return the result of the test
    return valid
# END DEF
```

Step 3: Write the subroutine getPositiveReal()

```
44  def getPositiveReal(message):
45      # SUBROUTINE  RETURNS real
46      #  LOCAL VARIABLES
47      #    valid   boolean   to indicate state of input
48      #    number  string    to hold the inputted number
49
50      # initialise switch for validity
51      valid = False
52      while not valid:
53          # get number
54          number = input(message)
55          # determine if the number is a real
56          if testForReal(number):
57              # now test number for being positive
58              if float(number) >= 0:
59                  valid = True
60              # end if
61          # end if
62
63          # determine if a message should be shown
64          if not valid:
65              displayMessage("invalid number, try again")
66              # END IF
67
68      # END WHILE
69
70      # return the positive real number
71      return float(number)
72  # END
```

Please note:

1. Make sure that you RETURN the number as a float – that is the data type it will need to be in the MAIN routine.

Step 4: Write the subroutine calculateHypotneuse()

This subroutine needs to use some mathematical subroutines that are found in PYHTON's `math` library. To use this, we `import` it. Add this line as the FIRST line of your code:

```
1  import math
2
```

Now, you can use the `pow()` and `sqrt()` functions to do the calculation. These need to be prefixed by the name of the library that they are in.

```
85  def calculateHypotenuse(a, b):
86      # SUBROUTINE   RETURNS real
87      #  PARAMETERS
88      #    a   real  to hold the length of side a
89      #    b   real  to hold the length of side b
90      #  LOCAL VARIABLES
91      #    c   real  to hold the length of the hypotenuse
92
93      # carry out the calculation
94      c = math.sqrt(math.pow(a, 2) + math.pow(b, 2))
95
96      return c
97
98  # END
```

- `pow()` is the function for powers and has two parameters – the number base and the exponent.
- `sqrt()` is the square root function and has one parameter – the number to find the square root of.

Step 5: Write the subroutine displayMessage()

```
94  def displayMessage(message):
95      # SUBROUTINE   RETURNS nothing
96      #  PARAMETERS
97      #    message  string  holds the string to output
98
99      # output message
100     print(message)
101
102 # END
```

Step 6: Write the subroutine displayReal()

This needs to ensure that the real number is shown to two decimal places.

This is done by setting up a format pattern and then applying it to the output.

The number of decimal places (`dp`) is passed in as a parameter and then it is used to construct a pattern:

```
{:0.3f}
```

This means that there will be 3 places after the decimal point and there will be no extra padding at the beginning of the number.

The code looks like this:

```
112 def displayReal(message, real, dp):
113     # SUBROUTINE   RETURNS nothing
114     #  PARAMETERS
115     #    message  string  holds the string to output
116     #    real     real    holds the number to output
117     #    dp       integer holds the value for the number
118     #                     of decimal places to show
119
120     # set up format
121     outputFormat = "{:0."+ str(dp) + "f}"
122
123     # output the value
124     print(message, outputFormat.format(real))
125
126 # END
```

Step 6: enable the program to run

```
128
129 # run the main program
130 main()
```

Things to note:

1. This line of code runs the main routine. Without it, the program will not run!

Step 7: Run your code and Test it

Testing the Program

Complete the Expected Result column.

a	b	Type of Test	Expected Result	Success
3	4	Real	c = 5	
120	140	Real		
18.3	33.5	Real		
0		Erroneous		
5	0	Erroneous		
a		Erroneous		
56	b	Erroneous		

Step 8: In Word:

1. Copy the entire code to a Word Document under the heading Task 6.
2. Draw a Test Plan table and put suitable tests into it
3. Carry out the tests and indicate on the table if they were successful or not.

Appendices

Appendix 1 : What is PYTHON?

Python is a widely used high-level programming language for general-purpose programming, created by Guido van Rossum and first released in 1991.

An interpreted language, Python has a design philosophy that emphasizes code readability (notably using whitespace indentation to delimit code blocks rather than curly brackets or keywords), and a syntax that allows programmers to express concepts in fewer lines of code than might be used in languages such as C++ or Java.

The language provides constructs intended to enable writing clear programs on both a small and large scale.

Python features a dynamic type system and automatic memory management and supports multiple programming paradigms, including object-oriented, imperative, functional programming, and procedural styles. It has a large and comprehensive standard library.

Python interpreters are available for many operating systems, allowing Python code to run on a wide variety of systems. CPython, the reference implementation of Python, is open source software and has a community-based development model, as do nearly all of its variant implementations. CPython is managed by the non-profit Python Software Foundation.

(from Wikipedia October, 2017)

Basic Syntax

The basic syntax in PYTHON is near to English.

There is a specific syntax that we will use and there are some rules for using them.

Input

pseudocode

```
INPUT identifier
```

PYTHON

```
identifier = input('prompt')
```

Output

Pseudocode

```
OUTPUT statement
```

PYTHON

```
print('message')
```

Selection

Pseudocode

```
IF condition THEN
  statement(s)
ELSEIF
  statement(s)
ELSE
  statement(s)
END IF
```

PYTHON

```
if condition:
  #statement(s)
elif:
  #statement(s)
else:
  #statement(s)
```

Conditional Statements

Conditional statements are those that result in either TRUE or FALSE. Operators are used to make comparisons.

operator	meaning	Example (a=7, b=4)
==	If the values of two operands are equal, then the condition becomes true.	(a == b) is not True.
!=	If values of two operands are not equal, then condition becomes true.	(a != b) is True.
>	If the value of left operand is greater than the value of right operand, then condition becomes true.	(a > b) is True.
<	If the value of left operand is less than the value of right operand, then condition becomes true.	(a < b) is not True.
>=	If the value of left operand is greater than or equal to the value of right operand, then condition becomes true.	(a >= b) is True.
<=	If the value of left operand is less than or equal to the value of right operand, then condition becomes true.	(a <= b) is True.

Chaining Conditions

Conditions can be chained together by using LOGICAL operators:

Operator	Description	Example (a=True, b=False)
and	Logical AND If both the operands are true then condition becomes true.	(a and b) is False.
or	Logical OR If any of the two operands are non-zero then condition becomes true.	(a or b) is True.
not	Logical NOT Used to reverse the logical state of its operand.	not (a and b) is True.

Iteration

Pseudocode

```
FOR counter, condition, step DO
  statement(s)
END FOR
```

PYTHON

```
for counter in range(1, 10):
  #statement(s)
```

Pseudocode

```
FOR EACH item IN List/Array/Set DO
  statement(s)
END FOR
```

PYTHON

```
for item in list:
  #statement(s)
```

Pseudocode

```
WHILE condition DO
  statement(s)
END WHILE
```

PYTHON

```
while condition:
  #statement(s)
```

Rules

1. identifiers must have meaningful names
2. identifiers must not have spaces and must use Camel Case
3. always input one value at a time
4. Lists, Arrays and Sets start with an index of 0

Reading list

Further Reading and Tasks

One of the fundamental programming tasks that you will learn about in this course is that of Web Programming.

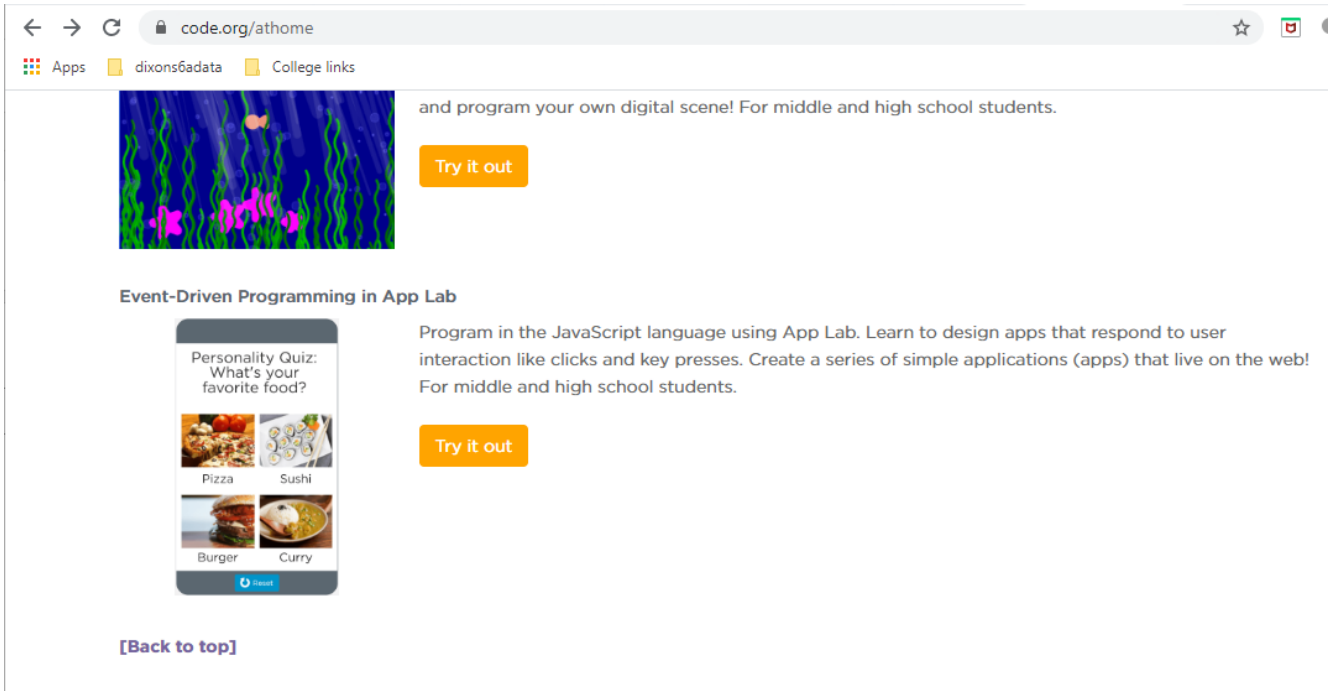
In order to prepare for that you could do this course on Code.org.

You will need to sign up, watch videos and then complete the tasks.

Keep a record of what you do by taking screenshots of your work and compiling it on a MS WORD page.

Step 1: Go to Code.org

Step 2: Find the course



You need to choose Event Driven Programming in App Lab

Step 3: Sign up to the website

Step 4: Complete the Course

You will find videos, tasks and much more in here.