Please refer to the online lesson plan on the DT Hub to access all website links and additional resources.

## Learning hook

Invite a mathematics teacher to the class and have them suggest ideas for simple programs the class can design that can be used by maths teachers for different topics in mathematics.

The mathematics teacher may also introduce some of the more interesting features of the Fibonacci series and the golden ratio.

Note: lesson sequence *Comparing and selecting appropriate algorithms: Fibonacci served three ways* uses the Fibonacci series extensively and the teacher may choose to teach this lesson first.

## Learning map and outcomes

The lesson illustrates the advantages of a modular approach to program design.

Students will develop individual modules which will be accessed by a single main program.

## Learning input

1. Explain to students the advantages of modules over functions.
(Using a modular approach breaks a complex program down into separate manageable parts forming related collections functions, classes, variables and other executable code.)
2. Explain how to create self-contained modules in the language with which students are familiar.
3. Explains how to access external modules from a main program.

**Simple Python example: modular approach**

Introduce the following example of the suggested modular approach. Use Python for illustrative purposes. The code is kept sparse and no formatting for output has been used. It does not use classes.

1. Introduce a mathematical module example.
	* The file name is the module name with the suffix .py appended.
	The following module (see file supplied) called fibo contains two functions: fibnums (which prints out the Fibonacci sequence up to a given value supplied by the user) and fibratio (which calculates the ratios of adjacent Fibonacci numbers up to a given value supplied by the user).
	* This module would be placed in the same directory/folder as all other modules produced by members of your class.
	* The module is saved as fibo.py

def fibratio(n):    # write Fibonacci ratios for pairs less than n
a, b = 0, 1
while b<n:
if a==0:
print()
print("Note that division by 0 is undefined")
print("First","Second","Fibonacci ratio")
else:
print (b,a,b/a)
a, b = b, a+b
print()
def fibnums(n): # return Fibonacci series up to n
result = []
a, b = 0, 1
while b < n:
result.append(b)
a, b = b, a+b
return result

1. Explain to students that the following module called tri\_num.py contains a function triangular\_number(n) which calculates all triangular numbers up to a given term.
This module should be placed in the same directory/folder as all other modules produced by the class.

def triangular\_number(n):
result=[]
for i in range(n):  # range(3) is a generator for [0, 1, 2]
n += i
result.append(n)
return result

1. Import all modules and their functions into the main program. Run the main module: this will need a menu of some kind for users to select the mathematical feature required.
In our example we have a main module we call main.py. Here is a possible implementation of main, showing a menu included inside a loop.

from fibo import \*
from tri\_num import \*
again="y"
while ((again =="y") or (again =="yes") or (again =="Yes") or (again =="YES")):
print()
print("Menu")
print("1 Fibonnaci ratios")
print("2 Fibonnaci numbers")
print("3 Triangular numbers")
menuchoice=int(input("Make a selection: type the menu item number "))
print()
if menuchoice==1:
        maxrat=int(input("print ratios of Fibonacci pairs less than what value? "))
        fibratio(maxrat)
    elif menuchoice==2:
        maxval=int(input("print Fibonacci numbers less than what value? "))
        print (fibnums(maxval))
    elif menuchoice==3:
        tri=int(input("print triangular numbers up to which term? "))
        print(triangular\_number(tri))
    again=input("Do you wish to choose again? ")
print("Farewell!")

## Learning construction

1. Organise students into groups of 3 or 4. Explains that each group will construct one maths module.
2. Lead the class in brainstorming possible ideas for mathematically based modules. They may be number or games based. The criterion is only that these could be useful for maths teachers to use with their students.
3. Ask each group to select a topic and write their module.

	1. Tell students to include three comment lines at the start of their module, followed by a brief explanation of its purpose.

# sample.py
# Student One, Student Two, Student Three
# May 24, 2016
# This program will…

* 1. Tell students to include an internal comment in their module specifying the purpose of the module and its input and output requirements. If using Python these could be in the form of a docstring.
	2. Ask students to create desk check tables to test their module. Approve each module prior to students uploading.
	<="" li="" style="box-sizing: border-box; font-family: Omnes-Regular, omnesregular, Arial; color: rgb(22, 53, 75);">
	3. Have students save their modules in a central location all groups can access.
1. Explain that each group is now required to create a main program, using their group’s name and including a menu which can be used to access all the modules. They should save this version locally in the same directory/folder with downloaded versions of other modules, to save any confusion between separate groups.

	1. An accompanying printed student worksheet may be designed by a student in the group to accompany the module.
2. Have students form groups to develop mathematically based modules using a suitable computer language with which students are familiar.

	1. An accompanying printed student worksheet may be designed by students to accompany the module.

**Some suggestions for possible modules**

* **Multiplication table generator**
* **A simple calculator**
* **Fibonacci numbers**
(Have students generate numbers using simple methods as well as recursion, finding golden ratios etc)
* **Factorials**
Students generate factorials – consider the use of recursion
* **Game of odds and evens**
Create chains starting from any integer: if even divide by 2, if odd multiply by 3 and add 1
* **Nim**
Ten counters in a row, the loser is the player who takes the last counter
* **Mastermind**
Suggest using 4 symbols with 4 positions as a text-based equivalent for this game with √ and X as feedback.
* **Lights-out**
A grid of two types of symbols representing either lights out or lights on. Lights adjacent to chosen one toggle to opposite state: aim is to have all turned on or all turned off
* **Tic-tac-toe/ Noughts and crosses**
* **Decimal to binary conversion**
* **Hexadecimal to decimal conversion**

## Learning demo

Have the groups demonstrate their final main program, showing that it can access other modules.

Ask each group to use their main module to access other modules and demonstrate it to the class.

## Curriculum links

| Links with Digital Technologies Curriculum Area |
| --- |
| **Strand** | **Content Description** |
| **Processes and Production Skills** | Implement, modify and debug modular programs, applying selected algorithms and data structures, including in an object-oriented programming language (AC9TDI10P09) . |

## Assessment

|  |  |  |
| --- | --- | --- |
|  | **Quantity of knowledge** | **Quality of understanding** |
| **Module development** | No evidence of understanding | Group develops correct algorithm | Group codes algorithm producing a working module | Group tests module and makes improvements | Group successfully improves another group's module (see Extension activity) |
| **Module integration** | No evidence of understanding | Group correctly uploads their module (working or not) | Group correctly integrates their module into main program (working or not) | Group integrates their module and other class modules into their own main program and creates working menu | Group successfully improves another group’s module and integrates it into their main program (see Extension activity) |
| **Optional Score** | 0 | 1 | 2 | 3 | 4 |