Pre-WA1 Practice Paper 1 (44 marks, recommended duration – 1 hour)

Answer **all** questions.

All tasks must be done in the computer laboratory. You are not allowed to bring in or take out any pieces of work or materials on paper or electronic media or in any other form.

Approved calculators are allowed.

Save each task as it is completed.

The use of built-in functions, where appropriate, is allowed for this paper unless stated otherwise.

The number of marks is given in brackets [ ] at the end of each question or part question.

Your program code and output for each of Task 1 to 3 should be saved in a single .ipynb file. For example, your program code and output for Task 1 should be saved as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

1 Name your Jupyter Notebook as

`TASK1_<your name>_<centre number>_<index number>.ipynb`

The task is to
- create user-defined functions
- perform simple operations on user-defined functions.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 1.1
         Program code

         Output:
```

Task 1.1

Write a function `product_two_integers(x, y)`.
The function will return the product of two integer values, `x` and `y`. [2]

Task 1.2

When an integer is divided by another integer, a quotient and a remainder is obtained.

For example, the division of 5 by 3 gives the quotient 1 and the remainder 2.

Write a function `quotient_two_integers(x, y)`.
The function will return the quotient of the division of integer `x` by integer `y`. [2]

Task 1.3

A function returns a single value which can be accessed using the assignment construct.

The pseudocode below shows how the return value of the function `test_A()` can be accessed when it is called within another function or proceudure `test_B()`.

```
FUNCTION test_B()

 k ← test_A() // return value of test_A() assigned to variable x

    …

ENDFUNCTION
```

Write a function `sum_product_quotient(x, y)`, which adds the product of two integers `x` and `y` to the quotient of `x` divided by `y`.

The function `sum_product_quotient` will make use of the return values of the functions:

- `product_two_integers` in **Task 1.1** to obtain the product of `x` and `y`
- `quotient_two_integers` in **Task 1.2** to obtain the quotient of `x` divided by `y`.

It will then return the addition of the return values as the required sum. [2]

Use `sum_product_quotient` to find the sum of the product of 55 and 45 and the quotient of 55 divided by 45. [1]

Task 1.4

Using an iterative construct, write a function `nth_power(base, exponent)` that will return the value of the expression $\texttt{base}^{\texttt{exponent}}$, where `base` and `exponent` are positive integers. [4]

Test your function with a suitable pair of arguments. [2]

Task 1.5

The digital root of a non-negative integer is the single-digit value obtained by an iterative process of summing digits, on each iteration using the result from the previous iteration to compute the digit sum. The process continues until a single-digit number is reached.

For example, the digital root of the number 987659876598765 can be obtain as follows:

9 + 8 + 7 + 6 + 5 + 9 + 8 + 7 + 6 + 5 + 9 + 8 + 7 + 6 + 5 = 105
1 + 0 + 5 = 6

Hence the digital root of 987659876598765 is 6.

Write a function `digital_root(n)` that:

- takes in a string value, `n` which contains only digits as its characters
- applies the algorithm described above
- returns the digital root. [8]

Save your Jupyter Notebook for Task 1.

2 Name your Jupyter Notebook as

`TASK2_<your name>_<centre number>_<index number>.ipynb`

The task is to create a standard deck of playing cards, shuffle these cards and distribute them to the players in a card game.

A standard deck of playing cards consists of 52 cards divided equally into one of four suits: clubs, diamonds, hearts or spades.

Each suit contains 13 cards made up of the numerical values 2 through 10 and the cards Jack, Queen, King and Ace.

For each of the sub-tasks, add a comment statement at the beginning of the code, using the hash symbol '#' to indicate the sub-task the program code belongs to, for example:

```
In [1] : # Task 2.1
         Program code

         Output:
```

Task 2.1

Write program code to create **two** lists, each containing 4 elements of the string data type.

The first list `suits` contains the following elements:

- `c` which represents the suit containing clubs
- `d` which represents the suit containing diamonds
- `h` which represents the suit containing hearts
- `s` which represents the suit containing spades. [1]

The second list `cards_1` contains the following elements:

- `J` which represents the cards containing Jack
- `Q` which represents the cards containing Queen
- `K` which represents the cards containing King
- `A` which represents the cards containing Ace. [1]

Task 2.2

Write program code to create a list `cards_2`, which contains 9 elements of the string data type.

The elements are the values 2 through 10, which represent cards containing these values.

You should use an iterative construct to create the list. [4]

Task 2.3

A playing card can be represented using a string containing two characters. The first character represents the suit that the card belongs to. The second character represents the value of the card.

For example:

- a card with value 5 that belongs to the suit containing hearts can be represented using the string `h5`
- a card containing the King which belongs to the suit containing spades can be represented using the string `sK`.

Write program code that makes use of the lists `suits`, `cards_1` and `cards_2` to create a new list `deck`.

The list `deck` contains 52 elements, each representing a card in the standard deck of playing cards. [5]

Output the list `deck`. [1]

Task 2.4

(For the purpose of completing this sub-task, if you were unable to create the list `deck` correctly in **Task 2.3**, you may do so by entering its elements into a list.)

The `random` module in Python contains a method `shuffle`, which can be used to shuffle (randomise the arrangement of) the elements of a mutable sequence.

The syntax for the method is `random.shuffle(seq, func)`, where:

- the parameter `seq` is a mutable data type
- the parameter `func` is an optional parameter which defaults to the function `random()`.

For this sub-task, the parameter `func` need not be specified.

Write program code to shuffle the elements in the list `deck`. [2]

Output the list `deck` again to show the sequence of values after shuffling. [1]

Task 2.5

A particular card game requires the cards to be evenly distributed to all the players. To do so, excess cards are removed from the top of the deck until the number of cards remaining is the largest possible multiple (equal to or less than 52) of the number of players.

For example:

- if there are 4 players, the largest possible multiple is 52; no card is removed
- if there are 5 players, the largest possible multiple is 50; two cards are removed.

Write a function `deal(players, shuffled_deck)` to simulate the following card dealing process.

Step 1 Determine if there are excess cards.
Step 2 Remove excess cards, if any.
Step 3 Deals the first player the card at the top of the deck by removing it and handing it to the player simultaneously.
Step 4 Proceeds to do likewise for subsequent players until every player is dealt with one card.
Step 5 Repeat Step 3 and Step 4 until all the cards have been dealt.

The function takes two parameters:

- `players`, a positive integer representing the number of players in the game
- `shuffled_deck`, a list representing a deck of standard playing cards that have been shuffled.

The function will return a nested list `dealt`, which represents the cards distributed to each player. [8]

Save your Jupyter Notebook for Task 2.