

## Part 1 (30 marks)

This part requires you to write some Python code to demonstrate your understanding of the basic coding constructions of the language. You should present evidence of your code working correctly in each case.

Write a function called `analyseWords()` that takes two parameters:

- A single mandatory positional argument: a list containing strings.
- A single optional keyword argument `minLetters` that can take an integer value greater than 0.

The function should calculate the average number of letters in the words in the list and return that average as a number that can then be displayed on the screen subject to the following rules:

- If `minLetters` is not specified, then all words should be considered in the calculation of the average.
- If `minLetters` is specified, then only those words that have at least that number of letters should be considered in the calculation of the average.
- If `minLetters` is specified incorrectly (e.g. set to -1), then a message should be displayed on the screen and the value 0 used instead.
- If the list parameter is empty i.e. `[]`, the returned value should be 0.

So, for example:

- A call `analyseWords(['dog', 'cat', 'emu'])` would return 3.
- A call `analyseWords(['dog', 'cat', 'emu'], minLetters=4)` would return 0.
- A call `analyseWords('flowers', 'of', 'romance')` would return 5.33.
- A call `analyseWords([])` would return 0.
- A call `analyseWords('flowers', 'of', 'romance'), minLetters=3)` would return 7.
- A call `analyseWords(['paris', 'in', 'springtime'], minLetters = -1)` would return 5.66 with a side message to note the invalid value of `minLetters`.

## Part 2 (30 marks)

This part requires you to write some Python code to solve a specific simple problem. You should consider what the variables and functions you may need to solve this problem. You should present evidence of your code working correctly.

Write a function `playGame(maxValue)` to perform the following task (it's based on an old school yard game called "Fuzz Buzz"):

Create a loop that depends on a variable `x` starting with the value 1 and increasing in value by 1 for each iteration of the loop, until `x` reaches a maximum value specified by the formal parameter `maxVal`. At each iteration of the loop the value of `x` should be displayed, unless one of the conditions shown below is true, when the message specified below should be displayed instead:

- If the value of `x` is divisible by 7, skip to the next value of `x` and display the word "skip".
- If the value of `x` is divisible by 9, display the word "fuzz"
- If the value of `x` is divisible by 4, display the word "buzz"
- If the value of `x` is divisible by both 9 and 4, display the words "fuzz" and "buzz", unless the first rule applies.
- Otherwise just display the value of `x`.

You can find out if a number is divisible by another using the modulo operator `%`. This tells you the remainder after integer division. For example:

8 % 2 equals 0 as 2 goes into 8 four times leaving no remainder.  
8 % 3 equals 2 as 3 goes into 8 two times leaving a remainder of 2.

You should then create a function `runGame()` that invites the user to enter an appropriate value for `maxValue`, and then calls the `playGame()` function accordingly.

### Part 3 (40 marks)

This part requires you to write some Python code using classes to solve a specific problem. Two classes are used to represent useful entities (parts of the problem that we want to solve). Solutions that do not incorporate a class solution will not be awarded the marks for this part of the assessment. You should present evidence of your code working correctly to solve the problem. To complete this part, you should use the `part3StarterCode` on StudentCentral. The starter code consists of two files with some code in them already:

- `VendingMachine`: a class to represent a vending machine at the University.
- `Item`: a class to represent a single item that is available for purchase from the vending machine.

The problem is concerned with a simple design for a vending machine. The program makes use of an `Item` class to represent an item that is available to buy, and `purchaseList` in the `VendingMachine` class to keep track of the items the user wishes to purchase. The file `VendingMachine.py` contains code to generate a variety of items for purchase. It then provides a very simple text-based User Interface (UI) that enables the user to:

- See a list of the items available for purchase.
- Select an item for purchase (i.e. add an item to their `purchaseList`)
- Remove an item from their `purchaseList`.
- See a list of all items in their `purchaseList`
- Get a price for all the items in their `purchaseList`
- Delete all items in their `purchaseList`.
- Check out their `purchaseList`. When the user checks out, the total is calculated. There is a discount of 5% for all orders over the value of £10.
- Quit the application

Your task is to create a working program, working from the starter code provided, to allow the user to do all the activities identified in the list above. The `Item` and `VendingMachine` files should contain the completed class definitions that provides all the functionality required to enable them to do their job.

### Incorporating code into your PDF report

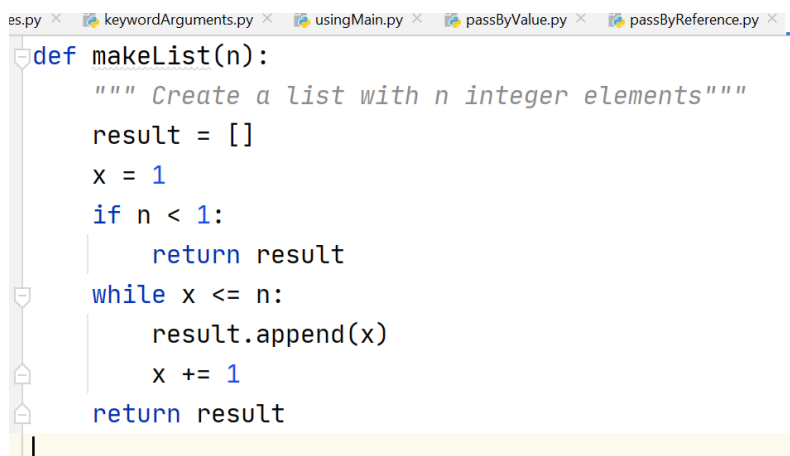
Your report must include the code that you produce and the evidence that the code is working. You can easily code the code from PyCharm into a Word document, but do take care to ensure that the code remains readable. Python relies on indentation, and the code in your report needs to retain the indentation to make sense. There are two easy ways of capturing your Python code in your report document:

- **Use a fixed width font:** something like Courier.
- **Use a screen shot tool:** typically “Snipping Tool” on PCs and “Grab” on Macs.

The result might look something like this:

```
def makeList(n):  
    """ Create a list with n integer elements"""  
    result = []  
    x = 1  
    if n < 1:  
        return result  
    while x <= n:  
        result.append(x)  
        x += 1  
    return result
```

Or this:



```
es.py × keywordArguments.py × usingMain.py × passByValue.py × passByReference.py ×  
def makeList(n):  
    """ Create a list with n integer elements"""  
    result = []  
    x = 1  
    if n < 1:  
        return result  
    while x <= n:  
        result.append(x)  
        x += 1  
    return result
```

For presenting evidence, using screenshots with appropriate supporting text commentary would be the easiest method.

## Marking guidance

For each part:

- 60% of the allocated marks will be for the code that you produce.
- 40% will be allocated for providing evidence of your code working.

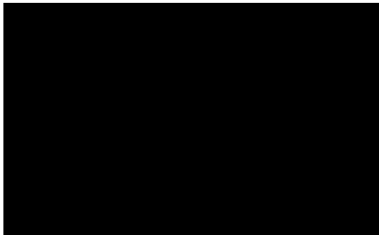
If your code does not work correctly, you can still get part credit for any parts that do work, and credit if you can explain what you think the problem is.

To get the best marks for the code part, you should consider:

- Use of appropriate variable and function/method names.
- Use of appropriate commenting and other code level documentation e.g. the Python docstring to help others understand what you code is about.
- Whether your code is reasonably succinct and easy to understand.
- Making good use of built in Python functions.
- Testing your code to ensure that you have taken reasonable steps to ensure that the program does not produce errors, and produces suitable messages if data provided to it does not make sense (i.e. user input errors).

To get the best marks for the evidence part, you should consider:

- Showing examples of your code working correctly and producing the correct output.
- Showing examples of your code producing appropriate error messages when there is a problem.
- Highlighting any areas where you think your code could be improved.



END.