**02: PYTHON BASICS CONTINUED**

**1: How many bisections do you expect to need if the tolerance is 10-3? 10-6? 10-9? Explain your reasoning.**

10 bisections would be expected if the tolerance is 10-3, 20 if the tolerance is 10-6 and 30 if the tolerance is 10-9. This is because the number of bisections is related logarithmically to the number of iterations. For every iteration, the range within which the root lies is halved. If the tolerance was one, there would be one iteration. If the tolerance was 1/10 there typically be 4 iterations; the range is split in half, then in quarters, then in eighths and finally in sixteenths. Once the range in question (one sixteenth) is smaller than the tolerance (one tenth), the iterations should stop. Thus, the number of iterations is found by finding the log2(1/tolerance). The answer is then rounded up to the next whole number to determine the expected number of iterations. log2(1/10-3) is equal to 9.96 (rounds up to ten). log2(1/10-6) is equal to 19.93 (rounds to 20) and log2(1/10-9) = 29.89 (rounds to 30). It is possible, by chance, to find the root using less bisections than the above formula predicts. However, you are guaranteed to find the root within the given tolerance using the above method.

**2: How (aside from actually changing the function in the code) could you make bisection.py easier to use for different functions?**

The function name f should be changed to something which tells the user something about what the function does.

More comments could be added to the code to make it easier for other users to follow the logic of the script and understand what each part does. However, for the most part the code is well-commented.

Variable names should be changed to describe what each variable actually corresponds to.

Only one function is used in the script. Using more functions could make it easier to use in other cases. The subsections of code which carry out different tasks could be put into functions, which could then be collected together in a module. For future applications, programmers would only need to import the module, type the function name and add the arguments, instead of typing out entire sections of code again.

Docstrings should be added for other users/programmers to get more information on the purpose and logic behind the function f should they need it.

**3: What are the differences between lists and tuples? Tuples and dictionaries? Sets and dictionaries? In what cases might you use each?**

Lists are mutable, tuples are immutable. Tuples are also faster to iterate through than lists.

Dictionaries contain key-value pairs; tuples contain elements. Each key can only appear once in a dictionary; tuples can contain duplicate elements. The order of the keys in a dictionary is irrelevant; the index of the elements is important. Dictionary values are mutable; tuples are immutable (elements cannot be changed).

Sets contain elements, while dictionaries contain key-value pairs. Different functions can be applied to sets, such as intersection, union and complement, which do not apply to dictionaries.

Tuples are used to contain elements that shouldn’t be changed; for example, the days of the week or the months of the year. Tuples are also used to store records, or pieces of information that relate to one another and that should be kept together. An example of this would be information relating to a person; his/her name, age, date of birth and occupation.

Dictionaries are used when the association between two things is key. For example, telephone directories, language dictionaries or functions used to determine how many times each letter occurs in a sentence all use dictionaries.

Sets are used a lot in mathematics and statistics. They normally store numbers or data in string form. The relationship between the data contained in two sets is normally what is of interest; finding overlap between data sets is the primary use of sets.

Lists are used in a wide variety of applications. Since they are easily mutable, any programme which involves iterating over a range of values and possibly adding/changing some values typically employs a list.

**4: How would you check within an if statement if a variable is a list or not? Why would you need to do this?**

The following code could be used to determine whether or not a variable (called “a” here) is a list or not:

if type(a) == type(list()):

print(“a is a list”)

else:

print(“a is not a list.”)

You would sometimes need to check if a variable corresponds to a list because some functions can only be applied to lists (and some other similar data structures). Examples of these functions include min(), max() and append(). If applied to a variable which wasn’t associated with a list, an Error would occur.