Sensible defaults, infinity and None - a note on Practical 3

Cast your mind back to the final question of Practical 3 where given a dictionary featuring island names as keys and their corresponding co-ordinates as lists, you were asked to find the smallest island. A suggested solution can be seen below in code snippet 1.

```
def smallest_island(islands_dict):

    # Set the minimum area found to 1,000,000 and the smallest island name to an empty string
    min_area = 1000000
    smallest = ""

    # Check each island in the dictionary of islands
    # Remember 'for key in dictionary' is a shortcut for 'for key in dictionary.keys()'
    for island in islands_dict:

        # Get the co-ordinates of the current island from the dictionary
        # and calculate the size with your landRectangleArea function
        coords = islands_dict[island]
        size = landRectangleArea(coords[0], coords[1], coords[2], coords[3])

        # If the current island size is smaller than the minimum area seen so far,
        # update min_area with the new minimum and update smallest to the current island name
        if size < min_area:
            min_area = size
            smallest = island

    # Once the loop is done, return the name of the smallest island
    return smallest
```

Code Snippet 1: A possible solution for Question 8 of Practical 3

Whilst this works, there was some debate as to a sensible default value for the variables `min_area` [1] and `smallest`. Hard-coding the minimum value seems a little arbitrary and could potentially introduce problems later on. What if one day your code was used to process much larger islands? If all islands had an area larger than 1,000,000 - the smallest island could not be identified (see the example in snippet 2).

```
>>> min_area = 1000000
>>> areas = [1000001, 2000000, 2500000, 3000000, 9999999]
>>> for area in areas:
...     if area < min_area:
...         min_area = area
...
>>> min_area
1000000 # Still 1,000,000 despite the smallest area in the list being 1,000,001

>>> min(areas)
1000001 # Built-in min function shows 1,000,001 as smallest element of areas list
```

Code Snippet 2: An example of the potential pitfall introduced by hard-coding `min_area`

[1] The example answer given on BlackBoard and in class actually used the variable name `min`, however this is a built-in function in Python for returning the smallest element in a list. Python is happy to let you 'overwrite' these functions without warning and care should be taken to avoid doing so. This is the same reason you shouldn't name your dictionaries `dict` or lists `list`. A list of built-in functions can be found at https://docs.python.org/2/library/functions.html.

Note that `min_area` is never updated to a new value as none of the entries in the `areas` list are smaller than our hard-coded default of 1,000,000.

We can't guarantee we could ever select an appropriate minimum value in this fashion, even if we chose a higher value as a default. What if somebody tries to use your program to calculate the smallest island given areas in small units like mm^2? Perhaps somebody else would try and use the code to handle island populations rather than area for very densely populated islands?

A suggestion from the class was to instead hard-code `min_area` to be the largest integer your computer could hold in memory. For most relatively modern desktops and laptops the largest `int` Python can store is 9,223,372,036,854,775,807. You can check this yourself as shown in snippet 3 using the `sys` package[2]; which you may recall having to import for writing to `stdout` and `stdin`. Along with that functionality, the package contains functions and variables that relate to your system - here we are interested in `maxsize`.

However, for reasons beyond the scope of this course it should be noted that Python is actually capable of storing numbers even larger than this and is only limited by the amount of memory available on your computer.

```
>>> import sys
>>> sys.maxsize
9223372036854775807
```

Code Snippet 3: Finding the largest positive integer supported by Python on your computer.

So the definition of "biggest number" is somewhat muddy here. Either way this feels almost as arbitrary as our hard-coded million, if not a little less open to the pitfall discussed. Surely there must be a better way?

You should by now be familiar with Python's built-in `int` and `float` functions, which you will have been using for converting strings from CSV files and user input to their numerical whole-number or decimal representations respectively. `float` has a special use case for creating floats that represent positive or negative infinity.

Snippet 4 replaces our hard-coded `min_area` with positive infinity.

```
def smallest_island(islands_dict):

    # Set the minimum area found to positive infinity and smallest island name to empty string
    # Any int or float will always be less than positive infinity
    min_area = float("inf")
    smallest = ""

    # Check each island in the dictionary of islands
    # Remember 'for key in dictionary' is a shortcut for 'for key in dictionary.keys()'
    for island in islands_dict:

        # Get the co-ordinates of the current island from the dictionary
        # and calculate the size with your landRectangleArea function
        coords = islands_dict[island]
        size = landRectangleArea(coords[0], coords[1], coords[2], coords[3])

        # If the current island size is smaller than the minimum area seen so far,
        # update min_area with the new minimum and update smallest to the current island name
        if size < min_area:
            min_area = size
            smallest = island

    # Once the loop is done, return the name of the smallest island
    return smallest
```

Code Snippet 4: Setting `min_area` to positive infinity.

[2]https://docs.python.org/2/library/sys.html

As any `int` or `float` will always be smaller than `inf`, here we can guarantee that whatever the input, there will never be an island area greater than our initial value for `min_area`, without the risk of setting the value to a large arbitrary number or the faff of querying the computer for the largest number it can hold in memory.

Similarly if we instead wanted to find the largest island, we could set a `max_area` variable to an initial value of negative infinity with `float("-inf")`. Any `int` or `float` will be greater than negative infinity.

For completeness, `float("infinity")` and `float("-infinity")` are also valid in both Python 2 and 3 and are semantically equal to `float("inf")` and `float("-inf")` but take longer to type.

* * *

Python also has a keyword called `None`, a special constant that represents the absence of value. Snippet 5 shows how we might use `None` in place of our hard-coded value.

```
def smallest_island(islands_dict):

    # Set the minimum area found to None and the smallest island name to an empty string
    min_area = None
    smallest = ""

    # Check each island in the dictionary of islands
    # Remember 'for key in dictionary' is a shortcut for 'for key in dictionary.keys()'
    for island in islands_dict:

        # Get the co-ordinates of the current island from the dictionary
        # and calculate the size with your landRectangleArea function
        coords = islands_dict[island]
        size = landRectangleArea(coords[0], coords[1], coords[2], coords[3])

        # Check whether min_area has a value set (it won't if this is the first island)
        if min_area == None:
            # If min_area is None, set it to be the current island
            min_area = size
            smallest = island
        elif size < min_area:
            # Otherwise, if min_area has a value, check whether the current
            # island is smaller than the minimum area seen so far and if so,
            # update min_area with the new minimum and update smallest to the
            # current island name
            min_area = size
            smallest = island

    # Once the loop is done, return the name of the smallest island
    return smallest
```

Code Snippet 5: Setting `min_area` to the `None` constant.

Note here that our `if` statement has changed to first check whether `min_area` is `None`, before trying to compare the current island's `size` to it. In Python 2, any variable: be it a `string`, `int`, `float` or even an empty `list` or `dict`, will be greater than `None`. Which means if we failed to set `min_area` to something other than `None` before beginning our comparisons, we'd still end up with a `min_area` of `None` at the end of the loop.
Yet, in Python 3, comparisons with `None` are a little more complicated and can return an error if the two types used in a comparison do not have a meaningful ordering [3].

As `min_area` is `None` initially, when the code processes its first island the `min_area == None` test will be `True` and the size of the first island will become the new `min_area` and `smallest` will be assigned its name.

All islands processed after the first will use the `elif` part of the statement (as `min_area == None` will now always be `False`) and work as before to compare the current island `size` to the `min_area` seen so far.

[3] Those interested can read a little more about this via https://docs.python.org/3/whatsnew/3.0.html#ordering-comparisons.