Flood Emergency Planning

Report for the CEGE0096: 2nd Assignment

Group: Gecko

Anastasia Kopytina
16005485
ucesopy@ucl.ac.uk

Daniella Harnett
20174920
ucesarn@ucl.ac.uk

Ziyou Huang
20052361
ucesuad@ucl.ac.uk

GitHub Repo Link: https://github.com/ana-kop/assignment_2.git

Introduction

The software developed for this assignment is part of the flood emergency planning for the Isle of Wight. The program asks the user to input their location and then finds the shortest route that they should take to walk to the highest point of land within a 5km radius; it then displays the map with the shortest route. The group has fully completed tasks 1-5, and attempted task 6; an extension was also made for creativity marks, outlined in the last section of this report.

The software developed as part of this assignment produces an operational solution to the problem, but it has a number of limitations.

- The program takes around 30 seconds to produce the final output. While the speed can certainly be improved, this was deemed to be acceptable for the purpose.
- The program is not able to show the user how to get from their location to the nearest ITN node, or from the ITN closest to highest point to the highest point itself.
- The program only displays the elevation on the final map if the user's location is within a box (430000, 80000) and (465000, 95000) – this is discussed further as part of task 6.
- The elevation raster data has low precision, meaning that the program sometimes identifies more than one highest point; the first point on the list is selected in these cases.
- The program only finds the route within 5km radius, with no option for the user to specify how far they are prepared to walk. The Naismith's rule also makes significant assumptions about the user's fitness.
- There is no option in the program to specify that a road has been flooded in order to calculate an alternative route. There is also no option to exclude the private roads.

The program was developed by writing some code from scratch using the lecture materials provided as part of the CEGE0096 module, and also by adapting some open-source code found in online communities.

It took around 2 weeks to develop a fully operational version of the program, and another week to debug and improve the code. The most challenging part of the assignment to write code for was task 6; unfortunately, the team was not able to provide a complete solution to the task, but only an attempt.

This software was completed through collaborative effort of the three group members. The tasks were divided between group members according to their abilities and interests, and regular meetings were then held to discuss problems. Git was used for version control and GitHub was used as the remote repository. Each team member created a separate branch to work on their task and then merged it into the master branch when the task was completed; afterwards all the branches apart from master were deleted. Overall, the work was split between group members in the following way:

- Anastasia: completed tasks 3-4 and the creativity task; responsible for the overall structuring of the program and project management.
- Daniella: completed tasks 2 and 5; worked on the technical aspect of task 6.
- Ziyou: completed task 1; worked on the conceptual aspect of task 6; responsible for the report.

Project Description

The software was developed using Python 3.8 in the PyCharm IDE, and so the program can be conveniently executed in this IDE. The program makes use of the following packages: json, networkx, rasterio, matplotlib, geopandas, shapely and numpy; the creativity task also uses pyproj and what3words. All the imports are already specified in the code.

The program requires the presence of the ‘Material’ folder (in the same format and layout as it was originally provided for this assignment) in the same directory as the main.py file to be executed in PyCharm. If the locations or the names of the input files have been changed, then these need to be updated in the following part of the code where the main() function is called.

```
if __name__ == "__main__":
    background_file = "Material/background/raster-50k_2724246.tif"
    elevation_file = "Material/elevation/SZ.asc"
    nodes_file = "Material/roads/nodes.shp"
    itn_file = "Material/itn/solent_itn.json"
    shape_file = "Material/shape/isle_of_wight.shp"

    main(background_file, elevation_file, nodes_file, itn_file, shape_file,
         island_checker=False, allow_near_edge=False, input_options=False)
```

Several options are presented in the main() function: allow_near_edge, island_checker, input_options. These are discussed in more detail in the sections devoted to tasks 6 and 7. For tasks 1-5 of the assignment, all three arguments need to be set to False.

Software description

Task 1: User Input

First of all, the user is asked to input their current location as a British National Grid coordinate (easting and northing). The program asks for each coordinate separately to ensure thorough error handling (input_options is an argument in the main() function that allows the use of extension in task 7).

```
if input_options is False:
      try:
          x = float(input("Please enter the Easting coordinate of your location: "))
      except ValueError:
          x = float(input("Please input a NUMBER. Please enter the Easting coordinate
      of your location: "))

      try:
          y = float(input("Please enter the Northing coordinate of your location: "))
      except ValueError:
          y = float(input("Please input a NUMBER. Please enter the Northing
      coordinate of your location: "))

      user_location = Point(x, y)
```

The coordinates are used to make a shapely point object. It is then used to test whether the user is within a box (430000, 80000) and (465000, 95000). This is done to avoid the 5km radius buffer being displayed incorrectly (this limitation is addressed in task 6). It is possible to turn off this additional requirement by setting the allow_near_edge argument in the main() function to True.

```
if allow_near_edge is False:
    if user_location.x <= 430000 or user_location.x >= 465000 or user_location.y <=
80000 or user_location.y >= 95000:
        print("Unable to assist in finding highest point of land.")
        exit()
    else:
        print("Input accepted, finding the shortest route to high ground…")
```

Task 2: Highest Point Identification:

Task 2 requires the developer to find the highest point in the 'elevation' ASCII file within 5km of the user location. This task was accomplished by firstly creating a 5km Shapely buffer around the user location. This buffer is then put into a GeoPandas Geodataframe (*buffer_gdf)* which is utilised by the mask module as shown below:

```
# Mask elevation with buffer polygon and crop; points outside of the buffer set to value of
-100
elevation_output_image, output_transform = rasterio.mask.mask(elevation,
buffer_gdf['geometry'], nodata=-100, crop=True, filled=True)
out_meta = elevation.meta  # Match the output image metadata with elevation metadata
out_meta.update({"driver": "GTiff",  # Update metadata for elevation output image
                     "height": elevation_output_image.shape[1],
                     "width": elevation_output_image.shape[2],
                     "transform": output_transform})
```

This section of code takes the elevation.asc file and masks the area outside of the buffer geometry to produce *elevation_output_image* which is cropped to the extent of the polygon. All values outside of the polygon are given a value of –100 which is used later for plotting. The output metadata is then matched to that of the original ASCII file and updated where necessary. The overall output is then written to a new file elevation*_output.tif.* This new file is then opened as *radius* and read in as a numpy array named *radius_arraay*. The highest point within this 5km radius is found using the *np.max* function. Once this value is found, it is necessary to find where this point is in British National Grid (BNG) coordinates. This is achieved using the following code:

```
pix_location_y, pix_location_x = np.where(radius_array == max_height)
```

The code above finds the array (pixel) location of the highest point (*max_height*) within the numpy array. Once the array coordinates have been found they can be converted to BNG coordinates using the rasterio transform module as shown below:

```
bng_pixel_location = radius.transform*(pix_location_x[i],pix_location_y[i])
```

Once transformed, *bng_pixel_location* holds a tuple containing the Easting/Northing coordinates of the highest point. These coordinates are then used to create a Shapely point for the next task.

Task 3: Nearest Integrated Transport Network

This part of the program identifies the nearest ITN node to the user and the nearest ITN node to the highest point identified in Task 2. First of all, the nodes.shp is read into a geodataframe.

```
nodes = gpd.read_file(nodes_f)
```

Next, for every node in the nodes geodataframe, its location (given as a shapely Point) is taken from the geometry column and, if it is less than 5km away from the user_location (i.e. if it is inside the 5km radius), the node is appended it to the nodes_list. A shapely MultiPoint object is then created from all the points in the nodes_list to allow the use of the shapely nearest_points method.

```
nodes_list = []
for i in range(len(nodes)):
    node_i = nodes.at[i, 'geometry']
    if node_i.distance(user_location) < 5000:
        nodes_list.append(node_i)

nodes_multipoint = MultiPoint(nodes_list)
```

Finally, a function nearest_itn is defined that takes location of a point of interest and a MultiPoint object containing the nodes, and returns the location of the ITN closest to the point of interest. This is done using the nearest_points method from shapely.ops. The method returns a tuple of points closest to the point of interest, so the first item in the tuple after the original point of interest is then selected and returned by the function.

```
def nearest_itn(point, multipoint):
    nearest_nodes_tup = shapely.ops.nearest_points(point, multipoint)
    nearest_node = nearest_nodes_tup[1]
    return nearest_node
```

The function is then used to get the coordinates of the closest ITN nodes to the user and the highest point respectively.

```
nearest_node_user_coord = nearest_itn(user_location, nodes_multipoint)
nearest_node_highground_coord = nearest_itn(highest_points[0], nodes_multipoint)
```

An additional step is also included in this part of the program, which is finding the FID of the ITN nodes obtained above. This is necessary since in task 4, the nodes in the graph are referenced by their FIDs when finding the shortest path between them. Firstly, a dictionary is made that allows mapping between the coordinate location of an ITN and its FID.

```
itn_fid_dict = {}
for i in nodes.index:
    geom = nodes['geometry'][i]
    fid_ = nodes['fid'][i]
    geom = str(geom.x) + ',' + str(geom.y)
    itn_fid_dict[geom] = fid_
```

Then a function coord_to_fid() is defined that takes coordinates of a given ITN node and returns its FID.

```
def coord_to_fid(itn_coord):
    point_to_convert = str(itn_coord.x) + ',' + str(itn_coord.y)
    res_fid = itn_fid_dict[point_to_convert]
    return res_fid
```

Finally, the ITN node coordinates are converted to their respective FIDs to be used in the next task.

```
nearest_node_user = coord_to_fid(nearest_node_user_coord)
nearest_node_highground = coord_to_fid(nearest_node_highground_coord)
```

Task 4: Shortest Path

In this part of the program the shortest route between the ITN node nearest to the user and the ITN node nearest to the highest point is identified using the Naismith's rule.

This task is completed using the graph functionality from the NetworkX package. Firstly, an empty MultiDiGraph structure is created. This particular type of graph is used since it allows multiple edges between one pair of nodes, and the edges are directed; this represents the ITN well, since any road can be used in two directions, and the time to cover it will not necessarily be equal in both directions due to differences in elevation. Next, the 'roadlinks' part of the solent_itn.json is loaded, to be used for creating the graph.

```
g = nx.MultiDiGraph()

with open(itn_f) as f:
    road_links = json.load(f)['roadlinks']
```

Next step is adding the edges and nodes to the graph. Every edge needs to have a weight which represents the time required to walk the respective road, calculated according to Naismith's rule. The formula to calculate this time is presented below (with time in minutes and distances in meters):

$$time = \frac{edge\ length}{\frac{250}{3}} + \frac{edge\ elevation}{10}$$

The first half of the equation calculates how fast a person will walk the length of the road, if the person is walking at 5km/h (or 250/3 m/min); the second half adds a minute to that time for every 10 meters that the person has to go upwards.

Firstly, a for-loop is initiated that iterates through every road link in the road_links dictionary, and finds the length of that road link and its coordinates.

```
for i, link_fid in enumerate(road_links):
    length = road_links[link_fid]['length']
    road_coords = road_links[link_fid]['coords']
```

The coordinates are then used to find the total elevation of the road, going in each direction. In order to do that, the elevation of each coordinate of a road link is read from the elevation raster using .sample() method (from rasterio); the elevation is then appended to the list road_coords_elev, in the same order as they appear.

```
road_coords_elev = []
for i, coord in enumerate(road_coords):
    x = coord[0]
    y = coord[1]
    elev = list(elevation.sample([(x, y)]))[0][0]
    road_coords_elev.append(elev)
```

Next, the total elevation of that road link is calculated from this list. This is done by taking elevation of two adjacent points on the road, and checking the elevation between them. If it is positive, it is added to the variable elev; at the end of the for-loop, this variable gives the total number of meters the person will have to go up when walking along this road.

```
elev = 0
for i in range(len(road_coords_elev) - 1):
    j = i + 1
    b = road_coords_elev[j]
    a = road_coords_elev[i]
    if b > a:
        elev = elev + (b - a)
```

Finally, the total time to cover that road is calculated, and used as the weight when adding the edge to the graph.

```
time_to_walk = ((3*length)/250) + (elev/10)
g.add_edge(road_links[link_fid]['start'], road_links[link_fid]['end'], fid=link_fid,
weight=time_to_walk)
```

Then, the road_coords_elev list is reversed to calculate the elevation of the road when it is being walked in the opposite direction, and the respective edge is added to the graph, with similar code being used as in the original direction.

Finally, after all edges have been added to the graph, the shortest path is calculated using the dijksta algorithm.

```
shortest_path = nx.dijkstra_path(g, nearest_node_user, nearest_node_highground,
weight="weight")
```

The code above returns the path as a list of FIDs, so these need to be converted into a LineString object before it can be plotted. Every pair of FIDs on the shortest_path list represent start and end nodes of a road. When pair of FIDs is passed to the get_edge_data method, it returns the FID of the road link. This can then be used to find the coordinates of the road. The coordinates of the road are then converted into a LineString object, which gets appended to a shortest_path_lines list. Finally, the LineStrings on the shortest_path_lines list are converted into a MultiLineString and then merged into one LineString to be used for plotting. While accessing the road link coordinates, the road length is also used to find the total path length to be displayed with the map to the user.

```
shortest_path_lines = []
path_length = 0
for i, c in enumerate(shortest_path[:-1]):
    fid = g.get_edge_data(c, shortest_path[i + 1])[0]['fid']
    coords = road_links[fid]['coords']
    shortest_path_lines.append(LineString(coords))
    path_length = path_length + road_links[fid]['length']
shortest_path_final = linemerge(MultiLineString(shortest_path_lines))
path_length = round(path_length / 1000, 2)
```

It is worth noting that in certain cases the shortest path may go outside of the 5km radius of the user's original location; an example of a case like this is shown in Figure 1 below (user location is 433600, 86600). This happens when there are no roads within the 5km radius that allow user to reach the closest high ground. (The high ground in these cases is still within the 5km radius of user's location.)

Furthermore, the integrated transport network on the Isle of Wight is quite sparse in certain areas, which means that the shortest path may end somewhere nearby the highest point, as for example, in Figure 1. In these cases, the user will most likely need to hike to the highest point using walking paths that are not part of the ITN. Similarly, if the user is in a rural area of the Isle to begin with, they may need to hike to the nearest ITN to begin walking on along the shortest path.

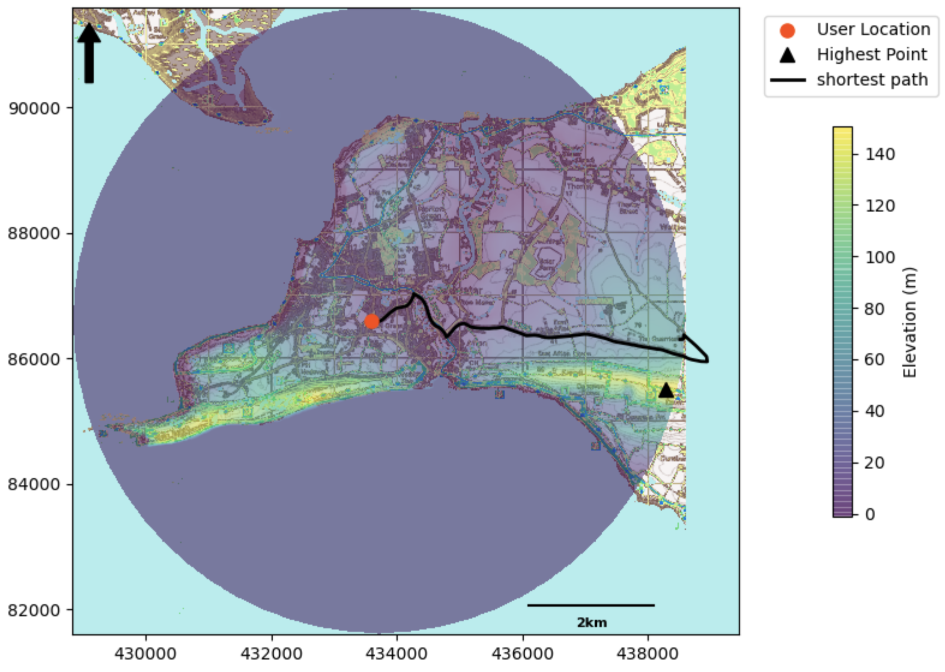

Figure 1

Task 5: Map Plotting

Task 5 required the combined outputs of the assignment to be plotted on a 10km x 10km map. For this task, the supplied TIFF file '*background*' was used but reduced to a 10km^2 window surrounding the user location. This was achieved by creating a Shapely bounding box with dimensions +/- 5km on the x and y axis from the user location as shown below:

```
# bounding box dimensions (for plotting the background file):
minx = location.x – 5000  #5km in each direction from user direction
maxx = location.x + 5000
miny = location.y – 5000
maxy = location.y + 5000
bbox = box(minx,miny,maxx, maxy)
```

This bounding box (*bbox*) was added to a geodataframe (*bbox_gdf*) which could be used by the following mask to crop the size of the background based on its geometry:

```
cropped_background_img,background_transform = rasterio.mask.mask(background, bbox_gdf[
'geometry'],crop=True)
```

Much like in Task 2, the output metadata was matched to the original file, updated and written to a new file called '*background_output*'. This new file is read in as *new_background* and its extent (required for plotting) is determined by the following:

```
background_extent = plotting_extent(new_background)
```

The numpy.ma module was used to mask the '-100' values from the 5km elevation data (from Task 2) and the '65' values from the background data as shown below:

```
circle = np.ma.masked_where(radius_array == -100, radius_array)  # Masking elevation
file to plot circular buffer
# Masking background sea level values to re-colour
removed_sea = np.ma.masked_where(cropped_background.read(1) == 65,
cropped_background.read(1))
```

Doing so prevents the elevation data filling the full extent of the map (allowing for it to be plotted as a circle) and removes the ocean values from the background, allowing them to be re-coloured separately in the final plot. Using matplotlib, the code used to plot the final figure is as follows:

```
fig, ax = plt.subplots(figsize=(8, 6))
ax.set_facecolor("paleturquoise")  # Set sea colour
ax.imshow(removed_sea, cmap="terrain", extent=background_extent)  # Plot cropped
background

if allow_near_edge is False:  # Task 6 extension: plot elevation if inside the box
    elevation_radius = ax.imshow(circle, alpha=0.5, aspect=1,
extent=background_extent, cmap="viridis")
    cbar = plt.colorbar(elevation_radius, orientation='vertical', fraction=0.025,
pad=0.12)
    cbar.set_label('Elevation (m)')

ax.plot(user_location.x, user_location.y, 'o', color='orangered', markersize=8,
label="User Location")
ax.plot(highest_points[0].x, highest_points[0].y, '^', color='black', markersize=8,
label="Highest Point")
plt.arrow((minx + 500), (maxy - 1200), 0, 950, width=120, head_length=300,
length_includes_head=True,
            facecolor="black", edgecolor="black")  # Plot north arrow
ax.plot([(maxx - 2500), (maxx - 500)], [(miny + 475), (miny + 475)], color="black",
        linewidth=1.5)  # Plot 2km scale bar
plt.text(maxx - 1750, miny + 125, "2km", fontsize=8, fontweight='bold')  # Plot scale
bar label

# Plot the shortest path
x, y = shortest_path_final.xy
ax.plot(x, y, color='black', linewidth=2, zorder=1, label="shortest path")

plt.legend(bbox_to_anchor=(1.025, 1), loc='upper left')
plt.tight_layout()  # Fit everything inside the figure
plt.rcParams.update({'figure.autolayout': True})
plt.show()
```

As the ocean is not the main focus of the user, the colour has been set to a consistent *pale turquoise.* The 10km x 10km background was added using '*.imshow*' on the existing axis with the colour map 'terrain' chosen for its applicability to the terrain of the region . Additionally, the extent of this plot is set to *background_extent* as determined previously. The elevation radius is overlaid with an alpha value of 0.5 to increase transparency and allow the user to use the base map. The elevation data uses the colour map *viridis.*

A variety of map elements are added to the final plot including a north arrow, scale bar, legend and elevation colour bar. The coordinates of the north arrow use the coordinates of the bounding box (which change depending on the users location). As shown, the arrow is offset from the bounding box by 500 on the x axis and 1200 from the top of the y axis. This ensures that regardless of the user location, the north arrow will maintain the same format. The scale bar is created similarly by generating a line (2km in length) using the coordinates of the bounding box. The legend and colour bar are located to the right of the map to prevent the elements becoming overcrowded.

Task 6: Extend the Region

Task 6 required the developer to write a code that overcame the limitation that the user must be more than 5km away from the edge of the elevation raster. The limitation was originally set in order to prevent the circular elevation buffer from being displayed incorrectly on the final map. This is because beyond the bounds of the raster file, there is no additional elevation data and therefore upon reaching this limit, the circular elevation buffer will be cut off. In many cases, the region cut off is the ocean which has little impact on the user, however since the project could be used in many environments, this limitation is acknowledged. The following code was proposed to overcome this challenge:

```
# create a padding 500 pixels wide around the elevation file filled with zeros
bigger_elevation = np.pad(elev_array, pad_width=500,
mode='constant', constant_values=0)
src_meta = elevation.meta
dst_transform = src_meta['transform']
out_meta_2 = elevation.meta       # match the output metadata with elevation metadata
out_meta_2.update({"driver": "GTiff",
                 "height": bigger_elevation.shape[0],
                 "width": bigger_elevation.shape[1],
                 "transform": dst_transform,
                 "crs": elevation.crs})
with rasterio.open(path+"/big_elevation_output.tif", "w", **out_meta_2) as dest:
    dest.write(bigger_elevation, indexes=1)
# write this new array to a tiff file that we can plot
big_elev = rasterio.open(path+"/big_elevation_output.tif", "r")
rasterio.plot.show(big_elev)
```

The output of this code segment can be seen in Figure 2. As shown, 500 pixels have been added as padding using the Numpy pad functionality. The new pixels were filled with zeros.

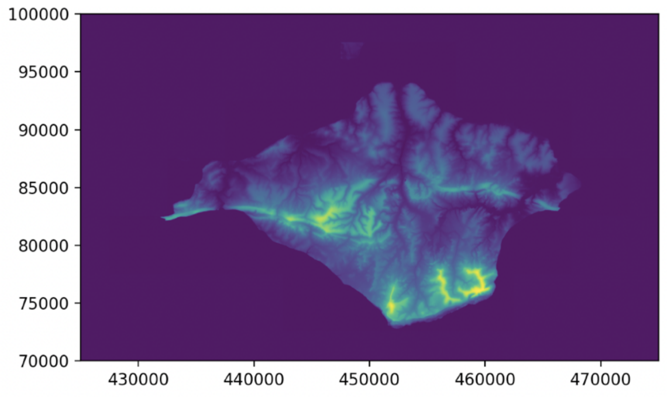

Figure 2

In principle, when combined with the rest of the software, this code would allow for a full 5km radius to be plotted around the user regardless of their location due to the extension of the file bounds. Unfortunately, by completing this transformation, the coordinates no longer match that of the original elevation file. Every effort was made to match the coordinates, for example the "crs" and "transform" objects are specified when 'writing' the bigger_elevation file however the team were not able to come up with an accurate enough solution. The decision was then made to use the original elevation file to preserve geographical accuracy which, during a flood event, would be of vital importance. Subsequently, the code presented above has not been included in the main.py.

Subsequently, a more simplistic solution is proposed. Firstly, an extension was written that allows the user to turn off the test written for task 1, so that user location is allowed to be outside the box (430000, 80000) and (465000, 95000). This is done by setting *allow_near_edge* argument to True in the main() function. In order to prevent the circular elevation buffer from being displayed incorrectly, when allow_near_edge is True, the circular elevation buffer does not get displayed. A message is also shown to notify the user of that. It was decided that while the circular elevation buffer is useful to display, the main purpose of the program is for the user to see the shortest path to the high ground, and that is still achieved.

Furthermore, in order to prevent the user from entering the location in the sea or somewhere else not on the Isle, an extension was written that checks whether user is on the Isle of Wight. This was done by loading the Isle of Wight boundary shapefile provided, then extracting the MultiPolygon object it contains and finally testing whether the user location point is contained within the MultiPolygon object; the program quits if the point is outside of the Isle of Wight. This extension can be turned on by setting *island_checker* argument to True in the main() function.

```
if island_checker is True:
    island_gpd = gpd.read_file(shape_f)  # Read in the Isle of Wight shapefile
    island = island_gpd['geometry'][0]  # Get the MultiPolygon object with the 
boundary
    if island.contains(user_location) is False:
        print('Looks like you are not on the Isle of Wight. '
              'This app is only intended to be used on the Isle of Wight.')
        exit()
```

Task 7: Creativity

An extension to the main code was created that gives user additional options for the format in which they input their location. The extension is enabled by setting input_options argument in the main function to True. The program will then give the user 3 options for how they would like to enter their location, and takes their response from input.

```
if input_options is True:
    option = float(input('How would you like to enter your coordinates? '
                         ' \nFor British National Grid, enter 1; for latitude and
                           longitude, enter 2; for what3words, enter 3: '))
```

Option 1 is the same as in the original version of Task 1.

Option 2 accepts user location as latitude and longitude. Pyproj Transformer is then used to convert the location to BNG.

```
transformer = Transformer.from_crs("EPSG:4326", "EPSG:27700", always_xy=True)
output = transformer.transform(lat_x, lon_y)
user_location = Point(output)
```

Option 3 allows user to input their location as what3words, which is a system that allows to identify a 3x3m location by a combination of 3 words. The what3words API is then used to convert location to latitude and longitude. An example of a location on the Isle of Wight is 'strike.resting.pythons'.

```
geocoder = what3words.Geocoder("JWCSJO0N")
res = geocoder.convert_to_coordinates(words)
x = res['square']['northeast']['lng']
y = res['square']['northeast']['lat']
output = transformer.transform(x, y)
user_location = Point(output)
```

References

1. GIS StackExchange, answer by Songololo on 5/01/2017, Weld individual line segments into one LineString using Shapely: https://gis.stackexchange.com/questions/223447/weld-individual-line-segments-into-one-linestring-using-shapely;
2. Lipani, Aldo, Course materials for CEGE0096;
3. Tenkanen, H., Nearest Neighbour Analysis: https://automating-gis-processes.github.io/2017/lessons/L3/nearest-neighbour.html;
4. What3words developer documentation: https://developer.what3words.com/tutorial/python

Git Log

```
commit 8c6682e2daf99f4c7a8513080c08649d6196c9bd (HEAD -> master, origin/master)
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Jan 11 09:22:43 2021 +0000

    Final commit

commit 6a3a0e8bfa721e731681b3c925d513100b0e6a44
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Jan 11 09:22:10 2021 +0000

    Final commit

commit 727a4e2ddeb9679a32a2d4c60f65a89c2b85609b
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sun Jan 10 19:22:08 2021 +0000

    Minor edits

commit f1c206342c23e87c69fd3fde4698d00f4e9927ed
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Jan 9 14:22:13 2021 +0000

    Minor edits

commit 69b5d214cbe8135fad90b4491f68a63bcddf0e7b
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Jan 9 13:35:29 2021 +0000

    Minor edits

commit df685fb7bef89f784371cc3014be966bafeb8826
Merge: ebe693c e7f6a6f
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Jan 9 10:44:44 2021 +0000

    Merge remote-tracking branch 'origin/master'

commit ebe693c6a5feafa3d0967ff1193d5b5b4a39742d
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Jan 9 10:44:17 2021 +0000
```

```
    Minor edits

commit e7f6a6f28cbebdccdbef58f3266750f4e2fb2c3b
Author: Yolanda-eng <ucesuad@ucl.ac.uk>
Date:   Thu Jan 7 17:08:16 2021 +0000

    Update main.py

commit 70eaf65574982dad321b1f98c754ed9e7b290433
Author: Yolanda-eng <73017949+Yolanda-eng@users.noreply.github.com>
Date:   Thu Jan 7 16:56:22 2021 +0000

    Delete task 1.py

commit 4bdbd781bec698a13d4c516b877aab1227cb8e41
Author: Yolanda-eng <73017949+Yolanda-eng@users.noreply.github.com>
Date:   Thu Jan 7 16:38:27 2021 +0000

    task1's code

commit 9b4c9709d78e912255b7292fe6f3fa2c0cf9f536
Author: ana-kop <ana.kop20@gmail.com>
Date:   Tue Jan 5 16:24:03 2021 +0000

    Added to main; creativity

commit ce818abd63e7cb9ac4f5741e81eac4473316ad11
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Jan 4 17:20:59 2021 +0000

    Improved path plotting

commit 1a30bd5b5ae1fd3184d1a1e084c37a891ff4a783
Author: ana-kop <ana.kop20@gmail.com>
Date:   Fri Jan 1 11:29:22 2021 +0000

    Minor edits

commit 0032262c156133a73eacbc9e40786d20b96f5703
Merge: 4a2e9ec 250c564
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Thu Dec 31 20:21:09 2020 +0000

    Removed duplicates.

commit 250c5648a2521ef06dddc139952bd5fdf1e297d8
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Thu Dec 31 19:10:50 2020 +0000

     Basic error handling added which ensures that the coordinates of the high point (transformed using
rasterio) return BNG coordinates that fall within the bounds of the elevation file. If returned coordinates
are inaccurate, the user may be using a file with a different CRS.

commit c4f14134ccfefecb0852b026f65d10584d36a5d1
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Wed Dec 30 18:21:09 2020 +0000
```

```
    PEP8 modifications made for task 2 and task 5

commit 4a2e9ec78b042d0786389b4fb1d7e0522af9ef80
Author: ana-kop <ana.kop20@gmail.com>
Date:   Tue Dec 29 18:18:50 2020 +0000

    Merging with branch 4

commit 14ff2f05804dcbd93ec9db53d8544b5b69175951
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Tue Dec 29 16:49:48 2020 +0000

    Changed variable names to become more understandable

commit a57796ddc91bd083e8f993049c889489203eba97
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Tue Dec 29 16:26:17 2020 +0000

    Altered masking of elevation radius to ensure it always plots the full circle. Tidied up comments a little.

commit 6719eb3e23bfc3fdb19a01714baf0dfa493bebce
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Tue Dec 22 18:37:22 2020 +0000

    added task 5 plotting user location, background, elevation radius, highest point, colour bar, scale bar, north arrow. Requires path. Combined tasks 1, 2 and 5.
    bold writing.

commit daedf5eaa9da174620d8d47547e785775798f2ff
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Tue Dec 22 18:28:28 2020 +0000

    added task 5 plotting user location, background, elevation radius, highest point, colour bar, scale bar, north arrow. Requires path. Combined tasks 1, 2 and 5.

commit 01ab3720c0369a7113b0719cc25e791f71b1b155
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Dec 14 19:12:26 2020 +0000

    Finding nearest ITN

commit 92798c9bee2c55487447f4b393c560224679a878
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Dec 14 18:30:27 2020 +0000

    Finding nearest ITN

commit 5b222774d25d256863e8f7ca59a3817795cbcc5c
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Dec 14 17:17:52 2020 +0000

    New branch

commit b5fbd5ddb67b6ddd19c7b63857b313b25bb9cf8f
Author: ana-kop <ana.kop20@gmail.com>
```

```
Date:   Mon Dec 14 17:04:05 2020 +0000

    Importing data

commit f109763dd35892ea74b75a5e6903012316f98c8a
Merge: f3a8a5a 621a93a
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Dec 14 17:00:18 2020 +0000

    Merge remote-tracking branch 'origin/master'

commit f3a8a5ab74c37b6d4b53ee2694c200bd93a25324
Author: ana-kop <ana.kop20@gmail.com>
Date:   Mon Dec 14 16:59:47 2020 +0000

    Importing data

commit 621a93a0d41c4b1c6eecca784aa3d834aeb851d4
Author: danni-harnett <72945718+danni-harnett@users.noreply.github.com>
Date:   Mon Dec 14 16:14:17 2020 +0000

    trial

commit bff65c7ed46835c479da999d4637f6959d069949
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Dec 12 13:53:45 2020 +0000

    First commit

commit bf26f12e57208d9266265b1d70d86d80a8c4ec08
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Dec 12 13:51:46 2020 +0000

    First commit

commit 4dad17c2955d65527e436e9f68202410a0141cce
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Dec 12 13:48:10 2020 +0000

    First commit

commit 1979a80fa6ad482430c58c0ebc4a5edf323d5c34
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Dec 12 13:45:54 2020 +0000

    First commit

commit 70474e8fb5681336f6cd6f1ebc41d11481036aaa
Author: ana-kop <ana.kop20@gmail.com>
Date:   Sat Dec 12 13:43:56 2020 +0000

    First commit
```