A comprehensive instruction for the basic Python programming practical test

Part 1: General

1 Code Explanation

Task: Explain the different components of a given Python code.

Steps:

1. Open the provided Python code. **c1001conditionElevationCreate.py**
2. Identify and explain the various components of the code (variables, functions, loops, etc.).
3. Describe the inputs and outputs of the code.

2 Debugging Code

Task: Correct and run a Python code with errors.

Steps:

1. Open the provided code with errors. **c1002 AddTwoNumbersWithUserInput.py**
2. Inspect the errors in the code.
3. Correct the errors and make the necessary corrections.
4. Run the corrected code to ensure it works properly.

Correct and run the next Python codes with errors. (repeat the above steps)

1. Open the provided code with errors. **c1003SampleInputOutputWithLoopError.py**
2. Inspect the errors in the code.
3. Correct the errors and make the necessary corrections.
4. Run the corrected code to ensure it works properly.

3 Writing Python Code

Task: Write Python code for given cases.

3.1 Case 1: Determining the UTM Zone of a Given Longitude

1. Write a simple Python code that loops through a list of values.
2. Determine whether each value falls within UTM zones 36, 37, or 38.

Test Instruction: Determining the UTM Zone of a Given Longitude

Objective:
To determine if a given longitude falls within UTM zones 36, 37, or 38 and to understand how to calculate the UTM zone from the longitude.
Steps:

1. **Input Longitude**: Enter a longitude in degrees, ranging from -180 to 180. The program will prompt you to enter this value.
2. **Calculate the UTM Zone**:
   - Add 180 to the longitude to shift the range to 0-360 degrees.
   - Divide the shifted longitude by 6.
   - Add 1 to the result to get the UTM zone.
3. **Check the UTM Zone**:
   - Compare the calculated UTM zone with 36, 37, and 38.
   - If the UTM zone matches any of these values, it is within the target range.

Example Tip:
For example, if the longitude is 42.0 degrees:

- Normalize the longitude: 42 + 180 = 222.
- Divide by 6: 222 / 6 = 37.
- Add 1: 37 + 1 = 38.
- The calculated UTM zone is 38, which is within the target range.

Solution: c1004utm_zone_checker.py

3.2 Case 2: Calculating Horizontal, Vertical, and Slope Distances Between Two Points

1. Write a code to calculate the horizontal, vertical, and slope distances between two given points.
2. Calculate the slope in percent.

Test Instruction: Calculating Horizontal, Vertical, and Slope Distances Between Two Points

- **Objective:**

To calculate the horizontal, vertical, and slope distances between two given points. Additionally, to calculate the slope in percent.

- **Steps:**

1. **Input Coordinates**: Enter the coordinates of two points, Point A (x1,y1)(x1, y1) and Point B (x2,y2)(x2, y2).
2. **Calculate Horizontal Distance**:
   - Subtract the x-coordinate of Point A from the x-coordinate of Point B.
3. **Calculate Vertical Distance**:
   - Subtract the y-coordinate of Point A from the y-coordinate of Point B.
4. **Calculate Slope Distance**:
   - Use the Pythagorean theorem to calculate the slope distance:

$$\text{slope\_distance} = \sqrt{(\text{horizontal\_distance})^2 + (\text{vertical\_distance})^2}$$

\

5. **Calculate Slope Percent**:
   - Calculate the slope in percent using the formula:

$$\text{slope\_percent} = \left(\frac{\text{vertical\_distance}}{\text{horizontal\_distance}}\right) \times 100$$

- **Example Tip:**

For example, if Point A is (3,4)(3, 4) and Point B is (7,10)(7, 10):

- Horizontal Distance: 7−3=47 - 3 = 4
- Vertical Distance: 10−4=610 - 4 = 6
-

$$\text{Slope Distance: } \sqrt{(4)^2 + (6)^2} = \sqrt{16+36} = \sqrt{52} \approx 7.21$$

$$\text{Slope Percent: } \left(\frac{6}{4}\right) \times 100 = 150\%$$

Solution: c1005distance_slope_calculator

3.3 Converting Hectare Values to Timad

Convert hectare values to local area measurement units, such as **timad**.

Test Instruction: Converting Hectare Values to Timad

Objective:
To convert hectare values to local area measurement units, specifically timad, where 1 hectare equals 0.5 timad.

Steps:

1. **Input Hectare Value**: Enter the value in hectares that you wish to convert to timad.
2. **Conversion Calculation**:
    - Multiply the hectare value by the conversion factor (0.5) to get the equivalent value in timad.
3. **Output the Result**: Display the converted value in timad.

Example Tip:
For example, if the value is 10 hectares:

- Conversion: 10×0.5=510 \times 0.5 = 5 timad

Solution: hectare_to_timad_converter.py

4 GUI Development

4.1 Task: Distance Calculator GUI

Steps: GIS Distance Calculator GUI

1. Open an existing GUI code. c1008UGIdistance_calculator_gui.py
2. Explain its components, purpose, inputs, and outputs.
3. Run the GUI code.

4.2 Task: Work with GUI codes with errors

Steps:

1. Load a GUI code with errors. **c1009elevation_difference_calculator_guierror.py**
2. Inspect the errors, correct them, and run the code.

Elevation Difference Calculator GUI

Objective:

To create a graphical user interface (GUI) application that calculates the elevation difference between two given elevation inputs.

Steps:

1. **Initialize the GUI**:
   - Use the tkinter library to create the main window and set the title and size of the window.
   - Add a menu bar with a "File" menu that includes an "Exit" option to close the application.
2. **Input Elevations**:
   - Add labels and entry fields for the user to input the two elevation values (Elevation 1 and Elevation 2).
3. **Calculate Elevation Difference**:
   - Add a button labeled "Calculate Difference" that, when clicked, calculates the difference between the two elevations.
4. **Display the Result**:
   - Use a message box to display the calculated elevation difference.

Example Tip:

For example, if the elevation values are Elevation 1 = 1500 meters and Elevation 2 = 1300 meters:

- Elevation Difference: 1500−1300= 200 meters

Solution: c1009elevation_difference_calculator_gui.py

4.3 Create A GUI for converting hectares to timad

Instructions:

Objective:
To convert hectare values to local area measurement units, specifically timad, where 1 hectare equals 0.5 timad, using a graphical user interface (GUI).

Steps:

1. **Create the GUI**:
   - Use the tkinter library to create a GUI window.
   - Add an entry field to input the value in hectares.
   - Add a button labeled "Convert" to trigger the conversion.
   - Add a label to display the result in timad.
2. **Input Hectare Value**:
   - Enter the value in hectares into the entry field.
3. **Conversion Calculation**:
   - When the "Convert" button is clicked, multiply the hectare value by the conversion factor (0.5) to get the equivalent value in timad.
4. **Output the Result**:
   - Display the converted value in timad on the GUI.

Example Tip:

For example, if the value is 10 hectares:

- Conversion: 10×0.5=510 \times 0.5 = 5 timad

Solution: c1010GUIhectare_to_timad_converter_gui.py

Part 5: ArcPy Integration

5 Opening and Running Existing ArcPy Code

Steps:

1. Load the existing ArcPy code. **A2024_01_reproject_spatial_data.py**
2. Explain its components (e.g., imports, functions, variables).
3. Run the code.
4. Load the input and output in ArcMap.
5. Change symbols and save the ArcMap project.

Answer:

Instructions for Trainees: Reprojecting Spatial Data with EPSG Code

Objective:

To reproject spatial data from one coordinate system to another using the EPSG code. This script takes an input dataset in WGS 1984 geographic coordinate system and projects it to UTM Zone 37N using the EPSG code 32637.

Steps:

1. **Specify Input and Output Paths**:
   - Define the paths for the input and output spatial datasets.
2. **Specify EPSG Code**:
   - Provide the EPSG code for the target coordinate system (e.g., 32637 for UTM Zone 37N).
3. **Create Spatial Reference**:
   - Create a spatial reference object using the EPSG code.
4. **Check for Existing Output Dataset**:
   - Check if the output dataset already exists and delete it if it does.
5. **Project the Data**:
   - Use the arcpy.Project_management function to reproject the data to the target coordinate system.
6. **Output Result**:
   - Print a completion message indicating that the reprojection is done.

Inputs:

1. **Input File Path** (InFile): Path to the input spatial data file in WGS 1984 geographic coordinate system.
2. **Output File Path** (outFile): Path where the reprojected spatial data will be saved.
3. **EPSG Code** (epsg_code): The EPSG code for the target coordinate system (e.g., 32637 for UTM Zone 37N).

Outputs:

- The script reprojects the spatial data and saves it to the specified output file path.
- **Example Tip:**

For example, if the input file path is C:/geodb2023/ethgeodb.gdb/admin4326/protectedareas and the EPSG code is 32637 for UTM Zone 37N, the script will reproject the spatial data and save it to C:/geodb2023/ethgeodb.gdb/admin32637/protectedareas32637

6 Reprojecting Multiple Feature Classes with EPSG Code

Explain the component of an existing Arcpy code

6.1 Case 1

1. **Load the Script**:

Open the file **A2024_02_reproject_multiple_feature_classes.py** in your preferred Python IDE or text editor.

2. **Inspect Components and explain**:
3. **Explain the Objectives**:
4. **List the Source of the Data**:
5. **Outputs and Their Locations**:
6. **Difference Between the Inputs**:

6.2 Case 2

1. **Load the Script**:

Open the file **A2024_03_CreaetDBreproject_clip_feature_classes.py** in your preferred Python IDE or text editor.

1. **Inspect Components and explain**:
2. **Explain the Objectives**:
3. **List the Source of the Data**:
4. **Outputs and Their Locations**:
5. **Difference Between the Inputs**:

7 Debugging ArcPy Code

Task: Open an ArcPy code with errors.

Steps:

1. Load the ArcPy code with errors. **A2024_04_arpyCreateDBerror.py**
2. Inspect and identify the errors.
3. Correct the errors.
4. Run the corrected code.
5. Load the input and output in ArcMap.
6. Change symbols and save the project.

Solution: A2024_04_arpyCreateDB,py

8 Creating a Model using ModelBuilder and generate python code

8.1 Models

Python can call almost all the tools in ArcToolbox and, in this way, repetitive processes can be automated. Before we begin writing scripts from scratch, we'll start with an example automatically generated by the ArcGIS ModelBuilder application.ModelBuilder is an application built into ArcCatalog (and ArcMap) that allows users to create a workflow visualization, called a *model*. Models not only visualizethe workflow, but can also be can be run to execute the workflow. ArcGIS Toolboxtools can also be run via ModelBuilder; Tools can be dragged into the model paneland connected to create the workflow. When a model runs, it executes tools and the underlying code statements that correspond to pieces of the model. The underlyingcode can be exported to a Python script and we can compare the workflow visualization with the code.

```
Follow steps 1-3 to create and export a simple model.
```

1) In ArcCatalog, to create a model like the one in Figure:

- Launch ModelBuilder from the button on the Standard toolbar.
- Open ArcToolbox with the ArcToolbox button on the Standard toolbar.
- Locate the Buffer (Analysis) tool in ArcToolbox (ArcToolbox →AnalysisTools →Proximity → Buffer)
- Click the Buffer tool and drag it into ModelBuilder from its toolbox in ArcToolbox. A rectangle labeled 'Buffer' will appear.
- Right-click on the new Buffer rectangle > Make variable > From Parameter >Input Features.
- Right-click on the Buffer rectangle (again) > Make variable > From Parameter > Distance.
- Double click on the Input Features oval and browse to:'C:/geodb/data/park.shp'.
- Double click on the Distance oval and set it to 100 meter.
- Right-click on the Input Features oval > rename 'inputFeatures'.
- Right-click on the Distance oval > rename 'distance'.
- Right-click on the Output Feature Class oval > rename 'outputFeatures'.
- Check that the result looks like the Figure.

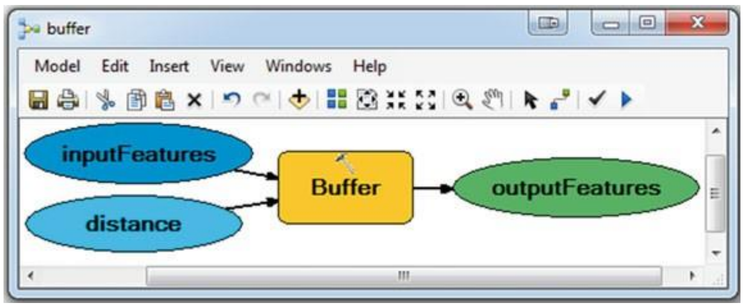
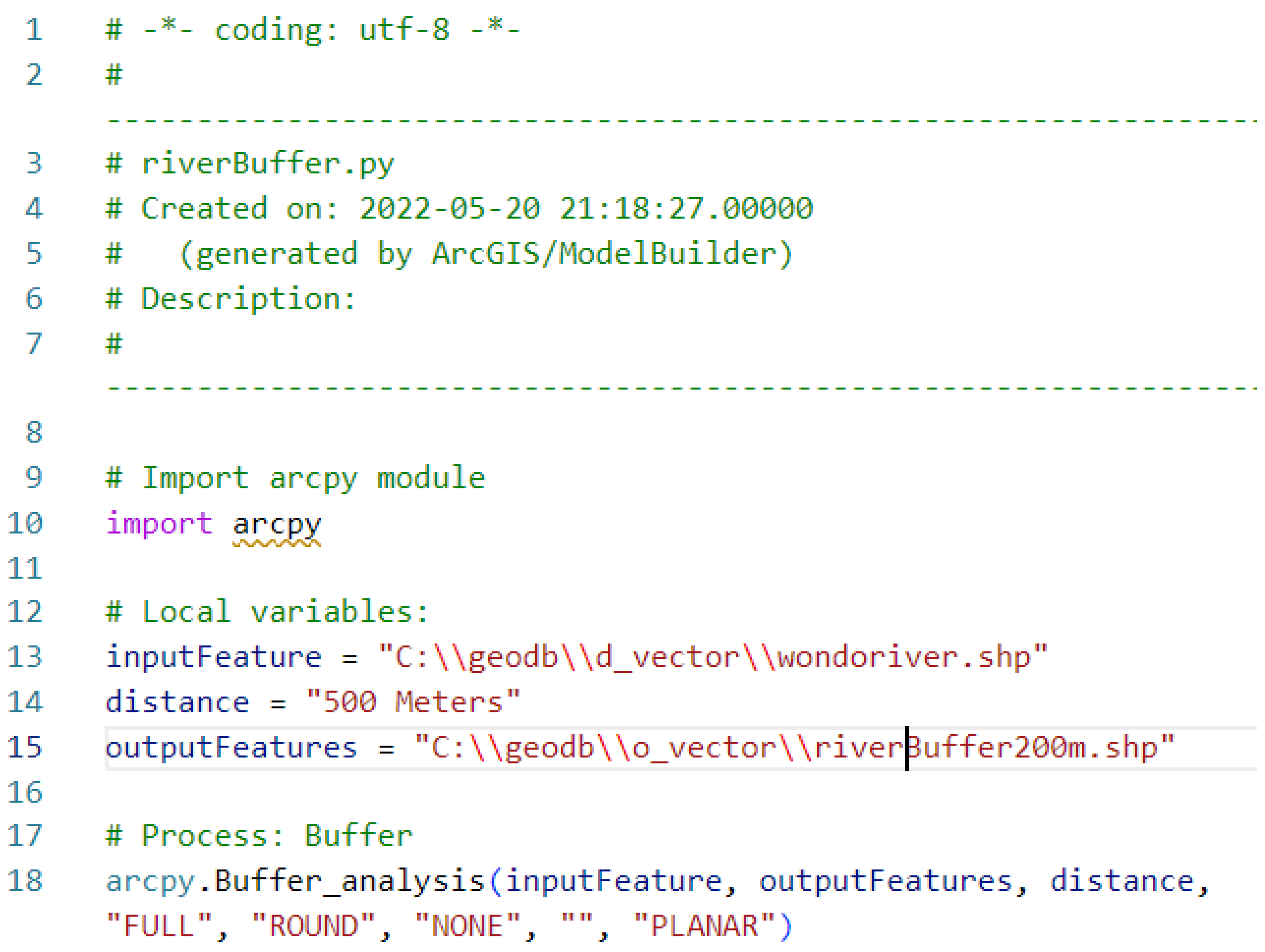


Figure: ModelBuilder Model to buffer input features.

2) Run the model to confirm that it works (Model menu > Run
3) Export the model as a script (Model > Export > Pythons Script). This should generate a script like figure below, shown with line numbers to the left of the code. Open the script in Visual Studio Code to view the code.



```
1   # -*- coding: utf-8 -*-
2   # ---------------------------------------------------------------
3   # riverBuffer.py
4   # Created on: 2022-05-20 21:18:27.00000
5   #   (generated by ArcGIS/ModelBuilder)
6   # Description:
7   # ---------------------------------------------------------------
8
9   # Import arcpy module
10  import arcpy
11
12  # Local variables:
13  inputFeature = "C:\\geodb\\d_vector\\wondoriver.shp"
14  distance = "500 Meters"
15  outputFeatures = "C:\\geodb\\o_vector\\riverBuffer200m.shp"
16
17  # Process: Buffer
18  arcpy.Buffer_analysis(inputFeature, outputFeatures, distance,
    "FULL", "ROUND", "NONE", "", "PLANAR")
```

Figure A script exported from the model shown above.

`To compare the model and script, we’ll look at each line of code.`

- Lines 1–7, 9, 12, and 17 are comments.
- Line 10 imports `arcpy`. This line of code enables the script to use ArcGIS commands. We’ll talk more about this in a moment.
- Lines 13–15 are assignment statements for string variables, `inputFeatures`, `distance`, and `outputFeatures`. These correspond to the model variables,the ovals that specify input and output for the tool. The string literal assigned tothe script variables depends on the value assigned in ModelBuilder. For example,on line 13, `inputFeatures` is being assigned the value ‘`inputFeatures = "C:\\geodb\\d_vector\\wondoriver.shp"` ’ because the model variable was given this value before the model was exported.
- Line 18 calls the Buffer (Analysis) tool. Running a tool in Python is like callinga function. We call the tool and pass arguments into it. The tool does our biddingand creates output or returns values. The variables and string literals in the parentheses are passing information to the tool. The Buffer tool requires three input parameters (the other parameters are optional). These required parameters are represented by the ovals in our model. The first three arguments in the Python correspond to these parameters. When the code was exported, it filled in the default values for the rest of the parameters. In summary, line 19 acts like the

rectangle in the model; it creates a buffer around the input features by the givendistance and saves the results in the output features.

With these observations, it's possible to get a feel for the connection between the model components and the lines of code in the script. In theory, exported models could be used as a starting point for scripts, but this approach can be cumbersome for several reasons. First, scripting enables a more flexible, reusable complex workflow, including functionality beyond ArcGIS geoprocessing. Second, exportedscripts usually require modification to perform as desired, making it more efficientto modify existing code samples than to build and export models.

The model/script comparison above provides some intuition for how the Pythoncode is working, though more detailed explanation is needed. The next sections address this need with discussions on importing `arcpy`, using dot notation with `arcpy`, and calling tools.

9 Describe existing arcpy code

10 Breakdown the Components of A2024_05_river_landcover_analysis.py

In your explanation, make sure to cover the following aspects of the code:

- **Objective**: What is the purpose of this script?
- **Inputs**: What data or files are required to run this script?
- **Outputs**: What results or files does this script generate?
- **Variables**: What are the key variables used in the script?
- **Processing Steps**: Outline the sequence of operations or steps the script performs.
- **Data Format**: What formats are the input and output data in?
- **Folders**: Where are the input files located and where are the output files saved?

This breakdown will help in understanding the structure and functionality of the script.

-

11 Writing ArcPy Code Without Loops

Task: Create a Python code for a selected case without using loops. Buffering the Land Cover Around All Springs

1. **Objective**: To create a buffer of 500 meters around all springs using the land cover and spring layers found under the data folder.
2. **Inputs**:
   - **Spring Shapefile**: pro_wondo2022\data\wondospring.shp
   - **Land Cover Shapefile**: pro_wondo2022\data\landcover2022.shp
3. **Output**:
   - A buffered shapefile with dissolved boundaries for all springs, named pro_wondo2022\data\landcoverspring_buffer_500m.shp

Solution: A2024_06_buffer_springs.py

12 ArcPy Code with Loops

Instructions for Buffering and Analyzing Land Cover Around Selected Springs

1. **Objective**:
   - To create a buffer of 500 and 1500 meters around the selected springs named Wondo SP, Wondo WK, and Bele using the spring and land cover layers found under the pro_wondo2022\data folder. Calculate the area of the land cover around these selected springs.
2. **Inputs**:
   - **Spring Shapefile**: pro_wondo2022\data\wondospring.shp
   - **Land Cover Shapefile**: pro_wondo2022\data\landcover2022.shp
   - **Field Name for Springs**: spname
   - **List of Springs**: Wondo SP, Wondo WK, Bele
   - **Distances**: 500 meters, 1500 meters
3. **Outputs**:
   - Buffered shapefiles with dissolved boundaries for selected springs.
   - Clipped land cover shapefiles around the selected springs.
   - Calculated area of the land cover around the selected springs.
4. **Python Filename**:
   - A2024_07_spring_landcover_analysisYourname.py

Solution: A2024_07_spring_landcover_analysis.py

13 NDVI Processing

13.1 Review the python code that handles raster data

Python Filename: `A2024_10_calculate_NDVI_2015.py`

Objective: Calculate the NDVI (Normalized Difference Vegetation Index) for Sentinel-2 imagery for the year 2015.

Task: Load the code and review its components. In your explanation address the following points:

- Objective:
- Inputs:
- Outputs:
- Variables:
- Processing Steps:
- Data Format:
- Folders:

Answer: Explanation:

Objective:
- To calculate the NDVI for Sentinel-2 images for the year 2015.

Inputs:
- Sentinel-2 imagery bands B8 (infrared) and B4 (red).

Outputs:
- NDVI raster file for the year 2015.

Variables:
- `year`: The year to process (2015).
- `imgINR`: Path to the infrared band image.
- `imgR`: Path to the red band image.
- `NDVI`: Calculated NDVI raster.
- `ndvi_file`: Path to save the NDVI raster.

Processing Steps:
1. Define the year to process.
2. Define paths to the infrared and red band images.
3. Calculate NDVI using the formula $(INR-R)/(INR+R)*100$(INR - R) / (INR + R) * 100.
4. Save the NDVI raster file.
5. Print completion message for 2015.
6. Print final completion message.

Data Format:
- Input: Raster images (GeoTIFF format).
- Output: Raster file (GeoTIFF format).

Folders:
- Input files are located in `C:/geodb2022/eth_central/image/`.
- Output files are saved in `C:/geodb2022/lc/`.

This script is focused on deriving the NDVI for the year 2015 only.

13.2 Create a python code that creates NDVI

Python Filename: `A2024_10_calculate_NDVI_2022.py`

Objective: Calculate the NDVI (Normalized Difference Vegetation Index) for Sentinel-2 imagery for the year 2022.

Task: Refer to the `A2024_10_calculate_NDVI_2015.py`

13.3 Evaluate and describe a python code that reclassify a raster layer e.g. NDVI

Open the code: A2024_12_reclassify_NDVI_2015.py

Answer

Objective:
- To reclassify the NDVI raster for Sentinel-2 imagery for the year 2015.

Inputs:
- NDVI raster file for the year 2015.

Outputs:
- Reclassified NDVI raster file for the year 2015.

Variables:
- year: The year to process (2015).
- ndvi_file: Path to the NDVI raster file.
- reclassified_file: Path to save the reclassified NDVI raster.

Processing Steps:
1. Define the year and file paths.
2. Check if the reclassified file already exists and delete it if necessary.
3. Reclassify the NDVI raster into land cover types.
4. Print completion message for 2015.
5. Print final completion message.

Data Format:
- Input: Raster file (GeoTIFF format).
- Output: Raster file (GeoTIFF format).

Folders:
- Input files are located in C:/geodb2022/lc/.
- Output files are saved in C:/geodb2022/lc/.

This script focuses on reclassifying the NDVI raster for the year 2015 only.

14 Edite the python cod that has errors

Purpose

Extract raster features by value based on user input for NDVI calculation.

Objective

Calculate and reclassify the NDVI (Normalized Difference Vegetation Index) for Sentinel-2 imagery for the years 2015 and 2022.

Code: A2024_13_calculate_reclassify_NDVIerror.py

Solution: A2024_13_calculate_reclassify_NDVI.py

Objective:

- To calculate the NDVI for Sentinel-2 images for the years 2015 and 2022.

Inputs:

- Sentinel-2 imagery bands B8 (infrared) and B4 (red).

Outputs:

- NDVI raster files and reclassified NDVI raster files indicating land cover types.

Variables:

- years: List of years to process.
- imgINR: Path to the infrared band image.
- imgR: Path to the red band image.
- NDVI: Calculated NDVI raster.
- ndvi_file: Path to save the NDVI raster.
- reclassified_file: Path to save the reclassified NDVI raster.

Processing Steps:

1. Loop through each year.
2. Define paths to the infrared and red band images.
3. Calculate NDVI using the formula $(IR-R)/(IR+R)*100$(IR - R) / (IR + R) * 100.
4. Save the NDVI raster file.
5. Reclassify the NDVI raster into land cover types.
6. Save the reclassified NDVI raster file.
7. Print completion message for each year.
8. Print final completion message.

Data Format:

- Input: Raster images (GeoTIFF format).
- Output: Raster files (GeoTIFF format).

Folders:

- Input files are located in C:/geodb2022/eth_central/image/.
- Output files are saved in C:/geodb2022/lc/.

This simplified version of the script retains the core functionality while being easier to understand and use.