

Problem Solving and Programming

Assignment 2: Self-Graded Answers for Deliverables 3, 5, 6, 7, 8

Compare your own submission using this answer sheet before the final exam.

Prior to the final exam, determine your mark for each part of each deliverable (based on these answers) and enter your self-assessment in the provided Assignment 2 Grading sheet. Total your self-assessment marks and enter it in the Assignment 2 Grading sheet. Change the name of the Assignment 2 Grading sheet to reflect your first and last names. Submit your Assignment 2 Grading sheet through D2L. All self-assessments must be submitted before the exam.

Please note: It is impossible to give every possible variation in the provided solutions. When you are completing your self-assessment, if your report contains code that you feel is valid and your code variation does not appear in the possible answers, please give a clear explanation in the comments indicating why you gave yourself that mark. If you are unsure, or if your code is completely different but achieves the same outcomes, then please talk to Karen.

General instructions that were part of Assignment 2 instructions: Include comments in your code, as described in class, and use meaningful variable names. When scripts are asked to be included in the report, copy and paste them into your report as text. Turn off automatic formatting. Ensure the indentation is appropriate for the interpretation of Python scripts. Only submit your individual Python source code files when specifically asked. Include all your screen captures in your report. Do not submit individual screen captures as separate files.

Deliverable 3: 6 marks

1. Submit the answers, in your report, to these questions for the *AddGeometryAttributes* tool:
 - a. What is the syntax? (not a screen capture)

```
AddGeometryAttributes(Input_Features, Geometry_Properties, {Length_Unit}, {Area_Unit}, {Coordinate_System})
```
 - b. What are the required parameters? Include a description (not a screen capture)

```
Input_Features:
```

 The feature layer to which the geometry properties will be added as new fields

```
Geometry_Properties:
```

 Specifies the geometry or shape properties that will be added to the feature layer
 - c. What are the optional parameters? Include a description (not a screen capture)

```
Length_Unit:
```

 units to be used for linear calculations

```
Area_Unit:
```

 units to be used for area calculations

```
Coordinate_System:
```

 coordinate system to use for calculating length, area and coordinates
 - d. Which parameters have defaults and what are their defaults? (not a screen capture)

```
Coordinate_System:
```

 coordinate system of the input features

Marking Scheme, maximum 2.4:

- a. **0.6** for correct syntax, showing required and optional parameters (0.6 each tool)
 - b. **0.6** for correct required parameters and a description (0.3 each parameter)
 - c. **0.9** for correct optional parameters and a description (0.3 each parameter)
 - d. **0.3** for indicating the 1 parameter with a default value and specifying it's default
2. Write a Python script in VS Code that uses the *AddGeometryAttributes* tool, to add the perimeter (in kilometers) and the centroid attributes to lakes.shp in your AutomationData folder

(Week10). Have the tool calculate the perimeters using the Canada Lambert Conformal Conic coordinate system. Include comments in the code. Submit a Screen Capture of the code and results in your report.

Version 1

```
# Deliverable 3
# Add the perimeter (in kilometers) and the centroid attributes to lakes.shp
# Calculate the perimeters using the Canada Lambert Conformal Conic coordinate system
import arcpy
import os
# cwd = os.getcwd()

# Set environment settings
arcpy.env.workspace = r"E:\Assignment2\AutomationData"
#arcpy.env.workspace = cwd + r"\AutomationData"

# Set local variables for use in AddGeometryAttributes tool
in_features = "lakes.shp"
properties = ["CENTROID"]

# Generate the centroid coordinates in the same coordinate system as lakes.shp using
# the Add Geometry Properties tool
arcpy.AddGeometryAttributes_management(in_features, properties)

# Create spatial reference (3 different ways)
sr = arcpy.SpatialReference(102002)
#sr = arcpy.SpatialReference("Canada Lambert Conformal Conic")
#sr = arcpy.SpatialReference(r"E:\Assignment2\CLCC.prj")
#sr = arcpy.SpatialReference(cwd + r"\CLCC.prj")

properties = ["PERIMETER_LENGTH_GEODESIC"]
length_unit = "KILOMETERS"
area_unit = ""
coordinate_system = sr

# Generate the perimeter in kilometers using Add Geometry Properties tool
arcpy.AddGeometryAttributes_management(in_features, properties, length_unit,
area_unit, coordinate_system)
```

Version 2

```
# Add the perimeter (in kilometers) and the centroid attributes to lakes.shp
# using the Canada Lambert Conformal Conic coordinate system
import arcpy
import os
cwd = os.getcwd()
# Set environment settings
arcpy.env.workspace = cwd + r"\AutomationData"

# Create spatial reference
sr = arcpy.SpatialReference(102002)
# Set local variables for use in AddGeometryAttributes tool
in_features = "lakes.shp"
properties = ["CENTROID", "PERIMETER_LENGTH_GEODESIC"]
length_unit = "KILOMETERS"
area_unit = ""
coordinate_system = sr

# Generate the centroid coordinates and perimeter in the
# Canada Lambert Conformal Conic using Add Geometry Properties tool
arcpy.AddGeometryAttributes_management(in_features, properties,
length_unit, area_unit, coordinate_system)
```

Marking Scheme, maximum 3.6:

Version 1 calculates the centroid with the same coordinate system as lakes.shp and the perimeter with Canada Lambert Conformal Conic. Version 2 calculates both the centroid and the perimeter with Canada Lambert Conformal Conic.

- a. **0.4** for importing arcpy and setting workspace (0.2 each)
- b. **0.6** for using variables with meaningful names for input features, properties, units, and coordinate system (0.15 each)
- c. **0.4** for specifying the properties in two lists, one list for the centroid and one list for the perimeter as in Version 1. **0.2** if you combined the properties into one list as in Version 2
- d. **0.4** for using the correct keywords for the centroid and perimeter properties with the perimeter being GEODESIC.
- e. **0.5** for specifying the length_units using the correct keyword (0.2 and for creating a spatial reference object using one of the three methods in Version 1 (0.3)
- f. **0.5** for using the AddGeometryAttributes tool **twice**, once for the centroid and once for the perimeter (as shown in Version 1) with all required and optional parameters specified in the correct order, including the length units and the spatial reference for the perimeter. **0.3** for using the AddGeometryAttributes tool **once** with the centroid and perimeter properties combined into the same list (as shown in Version 2)
- g. **0.3** for descriptive comments explaining your code
- h. **0.5** for the screen captures requested showing the code and results

Deliverable 5: 6 marks

- You have been supplied an ArcGIS Project file named Question5.aprx that uses the MapExampleData from Week 9's Map Scripting with ArcPy exercise. Place the MapExampleData from Week 9 in the same folder as the Question5.aprx and connect the layers in the map to the Europe Ecities and country feature classes in the countriesGDB.gdb geodatabase. The project file contains a layout
- Write a Python script that:
 - a. Moves the Ecities layer above the country layer
 - b. Adds the mjrivers feature class in the countriesGDB.gdb geodatabase to the map between the Ecities layer and the country layer. If you need to move it, then you may use insert to add the layer again into the correct position and delete the original layer.
 - c. Changes the symbology of the cities layer to use Equal Interval method, 4 classes and the POP_RANK field instead of the PORT_ID field, and for a bonus, a named cross symbol.
 - d. Changes the title to say "Population Ranks in Europe"
 - e. Moves the legend in line with the left side of the map
 - f. Adds your name to the credits
 - g. Exports the layout to a pdf with your login name followed by Europe

You may hard code the values for the variables

Include comments in the code, as described in class. Submit a Screen Capture of the code and results in your report.

Version 1 assumes all files are relative to the location of the python code file. Version 2 uses the os's current working directory for file paths. Version 3 hard codes the file paths.

```
# Deliverable 5
# Version 1
# Assumes the aprx file and the MappingExampleData folder are in the same folder
# as the python script file
# Assumes the script is being run from VS Code with the folder containing the aprx file
# open as the workspace
```

```

import arcpy

aprx = arcpy.mp.ArcGISProject(r"Question5.aprx")
fm = aprx.listMaps()[0]

# a. Moves the country layer below the Ecities layer
movlyr = fm.listLayers("Ecities")[0]
reflyr = fm.listLayers("country")[0]
fm.moveLayer(reflyr, movlyr, "BEFORE")

# b. Adds the mjrivers feature class in the countriesGDB.gdb geodatabase to the map
# between the Ecities layer and the country layer.
fm.addDataFromPath(r"MappingExampleData\countriesGDB.gdb\Europe\mjrivers")
movlyr = fm.listLayers("mjrivers")[0]
reflyr = fm.listLayers("country")[0]
# print(movlyr.name)
# print(reflyr.name)
fm.insertLayer(reflyr, movlyr)
# fm.moveLayer(reflyr, movlyr)
remlyr = fm.listLayers()[0]
fm.removeLayer(remlyr)

# c. Changes the symbology of the cities layer to use the Equal Interval method,
# 4 classes and the POP_RANK field instead of the PORT_ID field.
chnglyr = fm.listLayers("Ecities")[0]
# simple version without changing symbol
sym = chnglyr.symbology
if hasattr(sym, 'renderer'):
    if sym.renderer.type == "GraduatedColorsRenderer":
        sym.renderer.classificationField = "POP_RANK"
        sym.renderer.breakCount = 4
        sym.renderer.classificationMethod = "EqualInterval"
        sym.renderer.colorRamp=aprx.listColorRamps("Cyan to Purple")[0]
        chnglyr.symbology = sym

# d. Changes the title to say "Population Ranks in Europe"
# e. Moves the legend in line with the left side of the map
# f. Adds your name to the credits
lyt = aprx.listLayouts()[0]
elems = lyt.listElements()

for elm in elems:
    if elm.name == "Title":
        elm.text = "Population Ranks in Europe"
    elif elm.name == "Legend":
        mf = elm.mapFrame
        # moving the legend below the Data Frame left side
        elm.elementPositionX = mf.elementPositionX
    elif elm.name == "Credit":
        elm.text = elm.text + " Karen Whillans"

# g. Exports the layout to a pdf with your login name followed by Europe
lyt.exportToPDF(r"kwhillanEurope.pdf")

aprx.saveACopy(r"Question5modified.aprx")
del aprx, fm, movlyr, reflyr

```

```

# Deliverable 5
# Version 2
# Assumes the aprx file and the MappingExampleData folder are in the same folder
# as the python script file
# Uses cwd to get the operating systems current working directory
import arcpy
import os
from arcpy import env
cwd = os.getcwd()
env.workspace = cwd

aprx = arcpy.mp.ArcGISProject(cwd + r"\Question5.aprx")
fm = aprx.listMaps()[0]

# a. Moves the country layer below the Ecities layer
movlyr = fm.listLayers("Ecities")[0]
reflyr = fm.listLayers("country")[0]
fm.moveLayer(reflyr, movlyr, "BEFORE")

# b. Adds the mjrivers feature class in the countriesGDB.gdb geodatabase to the map
# between the Ecities layer and the country layer.
fm.addDataFromPath(cwd + r"\MappingExampleData\countriesGDB.gdb\Europe\mjrivers")
movlyr = fm.listLayers("mjrivers")[0]
reflyr = fm.listLayers("country")[0]
fm.moveLayer(reflyr, movlyr, "BEFORE")

# c. Changes the symbology of the cities layer to use the Equal Interval method,
# 4 classes and the POP_RANK field instead of the PORT_ID field.
chnglyr = fm.listLayers("Ecities")[0]
# simple version without changing symbol
sym = chnglyr.symbology
if hasattr(sym, 'renderer'):
    if sym.renderer.type == "GraduatedColorsRenderer":
        sym.renderer.classificationField = "POP_RANK"
        sym.renderer.breakCount = 4
        sym.renderer.classificationMethod = "EqualInterval"
        sym.renderer.colorRamp=aprx.listColorRamps("Cyan to Purple")[0]
        chnglyr.symbology = sym

lyt = aprx.listLayouts("Layout")[0]
# d. Changes the title to say "Population Ranks in Europe"
titleElem = lyt.listElements("TEXT_ELEMENT", "Title")[0]
oldsize = titleElem.elementWidth
titleElem.text = "Population Ranks in Europe"
# optional: move the title to a better position
newsize = titleElem.elementWidth
titleElem.elementPositionX = titleElem.elementPositionX + (oldsize - newsize)/2

# e. Moves the legend in line with the left side of the map
legendElem = lyt.listElements("MAPSURROUND_ELEMENT", "Legend")[0]
mf = legendElem.mapFrame
# moving the legend below the Data Frame left side
legendElem.elementPositionX = mf.elementPositionX

# f. Adds your name to the credits
creditElem = lyt.listElements("TEXT_ELEMENT", "Credit")[0]
creditElem.text = creditElem.text + " Karen Whillans"

```

```

# g. Exports the layout to a pdf with your login name followed by Europe
lyt.exportToPDF(cwd + r"\kwhillanEurope.pdf")

# create a new aprx file with the changes
aprx.saveACopy(cwd + r"\Question5modified.aprx")
del aprx, fm, movlyr, reflyr

# Deliverable 5
# Version 3
# Assumes the aprx file and the MappingExampleData folder are in the same folder
# as the python script file
import arcpy
from arcpy import env
env.workspace = r"E:\Assignment2"

aprx = arcpy.mp.ArcGISProject(r"E:\Assignment2\Question5.aprx")
fm = aprx.listMaps()[0]

# a. Moves the country layer below the Ecities layer
movlyr = fm.listLayers("Ecities")[0]
reflyr = fm.listLayers("country")[0]
fm.moveLayer(reflyr, movlyr, "BEFORE")

# b. Adds the mjrivers feature class in the countriesGDB.gdb geodatabase to the map
# between the Ecities layer and the country layer.
# Sometimes adds the data to the correct position, but not consistently
fm.addDataFromPath(r"E:\Assignment2\MappingExampleData\countriesGDB.gdb\Europe\mjrivers")

# c. Changes the symbology of the cities layer to use the Cross 1 symbol, Equal Interval
# method, 4 classes and the POP_RANK field instead of the PORT_ID field.
chnglyr = fm.listLayers("Ecities")[0]
# less simple version, includes bonus
sym = chnglyr.symbology
if hasattr(sym, 'renderer'):
    if sym.renderer.type == "GraduatedColorsRenderer":
        sym.updateRenderer("GraduatedSymbolsRenderer")
        chnglyr.symbology = sym

    #set symbol template
    sym = chnglyr.symbology
    symTemp = sym.renderer.symbolTemplate
    symTemp.applySymbolFromGallery('Cross 1')
    sym.renderer.updateSymbolTemplate(symTemp)
    chnglyr.symbology = sym

    sym = chnglyr.symbology
    # modify graduated colors renderer
    sym.renderer.classificationField = "POP_RANK"
    sym.renderer.breakCount = 4
    sym.renderer.classificationMethod = "EqualInterval"
    sym.renderer.colorRamp=aprx.listColorRamps("Cyan to Purple")[0]
    chnglyr.symbology = sym

lyt = aprx.listLayouts("Layout")[0]
# d. Changes the title to say "Population Ranks in Europe"
titleElem = lyt.listElements("TEXT_ELEMENT", "Title")[0]
oldsize = titleElem.elementWidth
titleElem.text = "Population Ranks in Europe"

```



```

# e. Moves the legend in line with the left side of the map
legendElem = lyt.listElements("MAPSURROUND_ELEMENT", "Legend")[0]
mf = legendElem.mapFrame
# moving the legend below the Data Frame left side
legendElem.elementPositionX = mf.elementPositionX

# f. Adds your name to the credits
creditElem = lyt.listElements("TEXT_ELEMENT", "Credit")[0]
creditElem.text = creditElem.text + " Karen Whillans"

# g. Exports the layout to a pdf with your login name followed by Europe
lyt.exportToPDF(r"E:\Assignment2\kwhillanEurope.pdf")

# create a new aprx file with the changes
aprx.saveACopy(r"E:\Assignment2\Question5modified.aprx")
del aprx, fm, movlyr, reflyr

```

Marking Scheme, maximum 6:

Version 1 assumes all files are relative to the location of the python code file. Version 2 uses the os's current working directory for file paths. Version 3 hard codes the file paths.

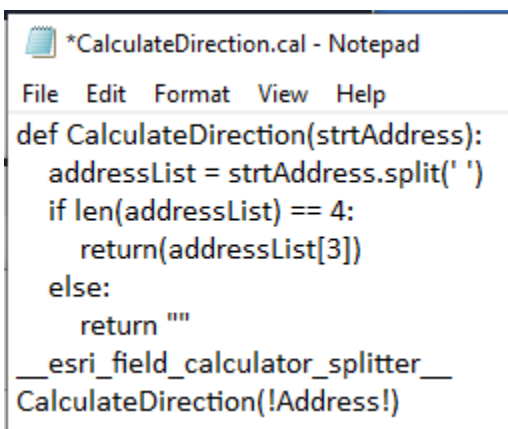
- a. **1** for importing arcpy, setting the environment workspace, getting a reference to the project file, and getting a reference to the map (0.2 each)
- b. **0.4** for getting references to the Ecities and country layers in the map (0.2 each)
- c. **0.4** for moving the Ecities layer above the country layer so Ecities is at the top and country is at the bottom of the table of contents (TOC) (0.2 moving, 0.2 order)
- d. **0.2** for getting a reference to the mjrivers feature class in the countriesGDB.gdb geodatabase and adding it to the map
- e. **0.3** for moving the mjrivers layer between the Ecities layer and the country layer (Versions 1 and 2 use different methods. Version 3 assumes the layer is added to the correct location based on geometry, which isn't always the case)
- f. **0.3** for checking that the symbology is a renderer of the type GraduatedColorsRender
- g. **0.6** for changing the symbology of the Ecities layer to use the POP_RANK field, 4 classes and Equal Interval method (0.2 each)
- h. **0.4** for finding the text element for the title and credits using their names (Versions 1 and 2 use different methods)
- i. **0.2** for changing the text of the title to say "Population Ranks in Europe"
- j. **0.2** for changing the text of the credits to include your name
- k. **0.2** for finding the map surround element for the legend using it's name (Versions 1 and 2 use different methods)
- l. **0.4** for moving the legend in line with the left side of the map (Versions 1 and 2 use different methods)
- m. **0.3** for exporting the map document to a pdf with the required name (0.2 export, 0.1 name)
- n. **0.3** for deleting the references to the project file, layers and map (0.1 each)
- o. **0.3** for descriptive comments explaining your code
- p. **0.5** for the screen captures requested showing the code and results
- q. **0.5 Bonus:** for changing the symbology of the cities layer to use a named cross symbol in as well as changing the field, number of classes and method (see Version 3)

Deliverable 6: 4 marks

Referring to the Python String Manipulation Lab exercise and using the same data (week 6), use Calculate Field in ArcGIS Pro to populate the "StrtDirect" field with the direction of each street address (eg. North). For those street addresses that do not have a direction, your code should assign an empty string (""). Export the working solution to a .cal file.

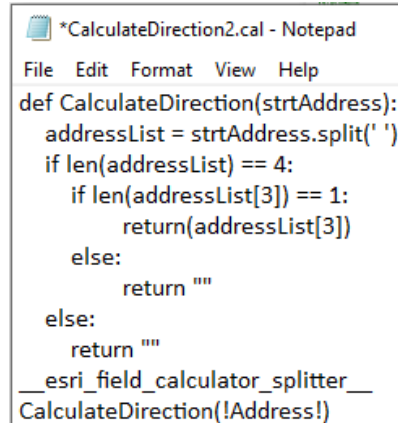
- Submit a screen capture of the contents of the .cal file and a screen capture of the Calculate Field window showing the code in the Code block and the Expression box.
- Include a screen capture of the Geoprocessing History window showing successful running of Calculate Field with your code.
- Also include a screen capture of the table, after running Calculate Field with your code, showing the "StrtDirect" field for several records in the table, including both records that have a direction and records that do not have a direction.

Version 1



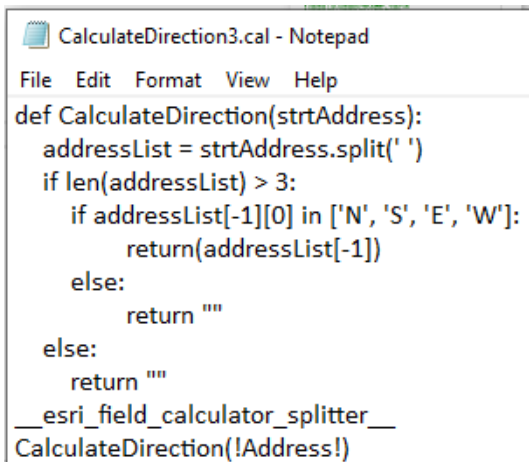
```
*CalculateDirection.cal - Notepad
File Edit Format View Help
def CalculateDirection(strtAddress):
    addressList = strtAddress.split(' ')
    if len(addressList) == 4:
        return(addressList[3])
    else:
        return ""
__esri_field_calculator_splitter__
CalculateDirection(!Address!)
```

Version 2



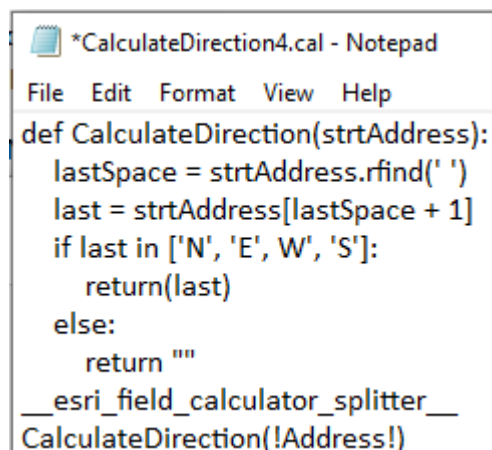
```
*CalculateDirection2.cal - Notepad
File Edit Format View Help
def CalculateDirection(strtAddress):
    addressList = strtAddress.split(' ')
    if len(addressList) == 4:
        if len(addressList[3]) == 1:
            return(addressList[3])
        else:
            return ""
    else:
        return ""
__esri_field_calculator_splitter__
CalculateDirection(!Address!)
```

Version 3

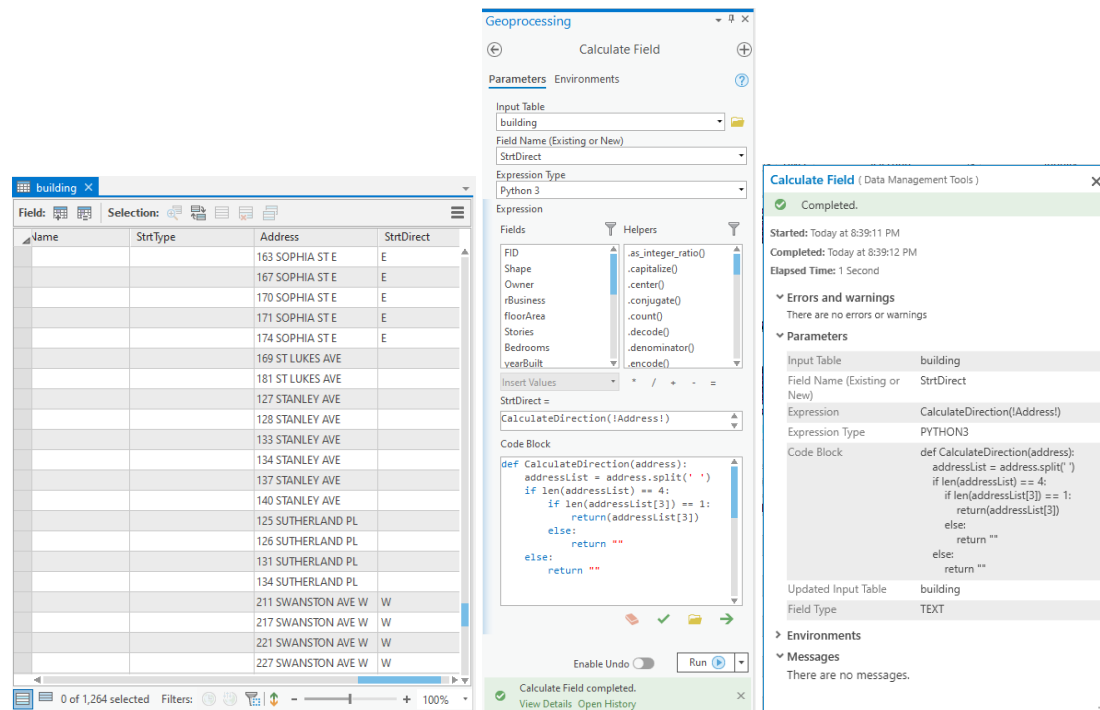


```
CalculateDirection3.cal - Notepad
File Edit Format View Help
def CalculateDirection(strtAddress):
    addressList = strtAddress.split(' ')
    if len(addressList) > 3:
        if addressList[-1][0] in ['N', 'S', 'E', 'W']:
            return(addressList[-1])
        else:
            return ""
    else:
        return ""
__esri_field_calculator_splitter__
CalculateDirection(!Address!)
```

Version 4



```
*CalculateDirection4.cal - Notepad
File Edit Format View Help
def CalculateDirection(strtAddress):
    lastSpace = strtAddress.rfind(' ')
    last = strtAddress[lastSpace + 1]
    if last in ['N', 'E', 'W', 'S']:
        return(last)
    else:
        return ""
__esri_field_calculator_splitter__
CalculateDirection(!Address!)
```

Marking Scheme, maximum 4:

Other string manipulation functions and a different structure to the code may achieve the same outcomes. Your code must do what the question asks but may use different string manipulation functions.

- 1.2** for screen captures requested showing successful running of the python code (cal file, Calculate Field window, geoprocessing results and table) (0.3 each)
- 0.6** for checking to see whether the Address string contains a direction
- 0.8** for obtaining the direction from the full Address string
- 0.6** for creating a function in the code block that **receives** the Address field as input and **returns** a string
- 0.4** for returning the direction or an empty string, as appropriate
- 0.4** for calling, in the expression box, the code block function using the Address field as an argument

Deliverable 7: 4 marks

- Add Python statements below each comment to sequentially do the tasks specified in the comments

```
# Program to display a menu of restaurant options and get the user's choice
# A dictionary of menu categories and options for each category will be used
# Task 1: Create a variable named category and assign the variable a list containing a
# minimum of 3 menu categories (breakfast, lunch, dinner, snack, ...)
```

```
category = ["breakfast", "lunch", "dinner", "snack", "beverage"]
```

```
# Task 2: Create a variable named categoryOptions and assign the variable
# lists of options for each category. (eg. the options for breakfast might be
# cereal, porridge, pancakes, bacon, eggs, ....). This variable will hold a list of lists.
# The number of options does not have to be the same for each category,
# but there should be more than one option for each category
```

```
categoryOptions = ["cereal", "porridge", "pancakes", "bacon", "sausage", "eggs"],
["sandwich", "chili", "falafel", "soup", "salad"], ["spaghetti", "burger", "stir fry"],
["chips", "fruit", "raw veggies", "dip", "cheese", "crackers"],
["milk", "tea", "coffee", "pop", "juice", "water"]]
```

Task 3: Create an empty dictionary

```
menu = {}
```

Task 4: Create a loop that uses the variable *category* to add the items in the
category list as keys to the dictionary and the items in the *categoryOptions* list as
values for each key (in the same order). For example, the key breakfast would have
a value that is a list containing cereal, porridge, pancakes, bacon, eggs

Version 1

```
for item in range(len(category)):
    menu[category[item]] = categoryOptions[item]
```

Version 2

```
for item in range(len(category)):
    one_category = category[item]
    optionsList = categoryOptions[item]
    menu[one_category] = optionsList
```

Task 5: Use a loop and the dictionary to display the menu categories

Version 1

```
print("Welcome to Fleming cafe")
print("Available menu categories are:")
for item in menu.keys():
    print(item)
```

Version 2

```
print("Welcome to Fleming cafe")
print("Available menu categories are:")
categoryList = menu.keys()
for item in categoryList:
    print(item)
```

Task 6: Ask the user which category they would like to order from and get their

reply. Display the user's choice of menu category

```
print()
categoryChoice = input("Which menu category would you like to order from? ")
print("You have chosen to order from", categoryChoice)
```

Task 7: Based on the user's choice of menu category, use a loop and the dictionary

to display the category options for the user's choice

Version 1

```
print()
print("For", categoryChoice)
print("Available options are:")
for item in menu[categoryChoice]:
    print(item)
```

Version 2

```
print()
print("For", categoryChoice)
print("Available options are:")
options = menu[categoryChoice]
for item in options:
    print(item)
```

Task 8: Ask the user which category option they would like to order and get their

reply. Display the user's choice of category option

```
print()
optionChoice = input("Which", categoryChoice, "option would you like to order? ")
print("You have chosen to order", optionChoice)
```

- Submit a Screen Capture of the code and results in your report.
- **Bonus:** Modify this program to add the cost of each category option to the dictionary as a list and once the user chooses an option, display the cost for that option

```
# Change to Task 2
categoryOptions = [[["cereal", "porridge", "pancakes", "bacon", "sausage", "eggs"],
["1.50", "2.00", "3.70", "4.56", "5.23", "2.15"]],
[["sandwich", "chili", "falafel", "soup", "salad"],
["10.30", "15.80", "8.30", "5.60", "6.20"]],
[["spagetti", "burger", "stir fry"], ["14.80", "9.40", "16.10"]],
[["chips", "fruit", "raw veggies", "dip", "cheese", "crackers"],
["1.50", "2.30", "3.70", "1.30", "2.50", "1.30"]],
[["milk", "tea", "coffee", "pop", "juice", "water"],
["1.60", "1.20", "1.40", "2.00", "2.50", "1.00"]]]

# Change to Task 7
# Display category options
print()
print("For", categoryChoice)
print("Available options are:")
for item in menu[categoryChoice][0]:
    print(item)

# Addition to Task 8
# Get cost of users choice of category option
costPosition = menu[categoryChoice][0].index(optionChoice)
print("Your order will cost $", menu[categoryChoice][1][costPosition])
```

Marking Scheme, maximum 4:

- 0.3** each for Tasks 1, 6 and 8. Categories, options and wording may be different.
- 0.2** for Task 3. Must use {} but dictionary name may be different
- 0.4** for Task 2. Must have a list for each category inside one outer list
- 0.6** for Task 4. Task 4 must use a for loop with the range function to create index values. (0.2)
It must retrieve an item from the category list and an item from the categoryOptions list using the index value (0.2). It must add the item from the categoryOptions list to the menu using the item from the category list as a key (0.2).
- 0.6** for Task 5. Task 5 must get the dictionary keys (0.3). It must use a for loop with the dictionary's keys to display the dictionary keys (0.3).
- 0.6** for Task 7. Task 7 must retrieve the list of options for the category chosen by the user from the dictionary (0.3). It must use a for loop with the list of options to display each option (0.3).
- 0.2** displaying meaningful messages for the user
- 0.5** for the screen captures requested showing the code and results
- 0.5 bonus:** for making changes to Tasks 2, 7 and 8 to store the price of each option in the dictionary as a list, display the category options (obtained from the dictionary) and display the price for the user's option choice (obtained from the dictionary)

Deliverable 8: 4 marks

1. The following code segment creates a Lake class

```
class Lake(object):
    def __init__(self):
        self.name = name

    def SDIndex(self):
```

1. Modify the Lake class definition as instructed below

- Add an area property and a perimeter property to the Lake class
- Allow the lake object to be given a name, area and perimeter when the Lake object is created (instantiated).
- Complete the SDIndex method to calculate and return the shoreline development index for the lake. The method should use the area and perimeter properties of the lake object. The formula for calculating shoreline development index is

$$\frac{L}{2(\sqrt{\pi A})}$$

where L = shoreline length (perimeter)

A = lake surface area

Version 1

```
import math
class Lake(object):
    def __init__(self, name, area, perimeter):
        self.name = name
        self.area = float(area)
        self.perimeter = float(perimeter)

    def SDIndex(self):
        sdi = self.perimeter / (2 * (math.sqrt(math.pi / self.area)))
        return sdi
```

Version 2

```
import math
class Lake(object):
    def __init__(self, name, area, perimeter):
        self.name = name
        self.area = float(area)
        self.perimeter = float(perimeter)

    def SDIndex(self):
        if self.area > 0:
            return self.perimeter / (2 * (math.sqrt(math.pi / self.area)))
        else:
            return 0
```

Marking Scheme, maximum 2.0:

- 0.2** for including the area and perimeter as parameters for the `__init__` function
- 0.5** for giving the lake object a name, an area and a perimeter when created
- 0.2** for converting the area and perimeter to a numeric data type
- 0.2** for importing math which is required for pi
- 0.4** for using the lake object (self) with area and perimeter to calculate the shoreline development index
- 0.2** for returning the calculated shoreline development index

- g. **0.3** for checking to make sure the area is not zero before doing the calculation and taking appropriate action when the area is zero. The SDIndex method must return something, either the calculated shoreline development index or a number that indicates there was a problem (must be a value below 1)
2. Create a short Python program that does the following, in the order listed
- a. Obtain's the lake's name, area and perimeter from the user
 - b. Create's the lake object with the name specified by the user and assigns the lake object the surface area and perimeter specified by the user
 - c. Determines the lake's shoreline development index using the SDIndex method
 - d. Determines whether the lake is a round lake or not. A round lake has a shoreline development index of 1.
 - e. Displays the lake's name, shoreline development index and whether it is a round lake or not

Submit a Screen Capture of the code and results in your report.

Version 1

```
from LakeClass import Lake
lakeName = input("What is the name of your lake? ")
lakeArea = float(input("What is the area of your lake? "))
lakePerimeter = float(input("What is the perimeter of your lake? "))

myLake = Lake(lakeName, lakeArea, lakePerimeter)
lake_sdi = myLake.SDIndex()
print(myLake.name, "has area", myLake.area, "and perimeter", myLake.perimeter)
print(myLake.name, "has a shoreline development index of", lake_sdi)
if lake_sdi < 1:
    print("Error with shoreline development index.")
elif lake_sdi > 1:
    print("This lake is not perfectly round")
else:
    print("This lake is perfectly round")
```

Marking Scheme, maximum 2.0:

- a. **0.1** for importing the lake class, if the class definition is in a different file from the application code. Must use the name of the class file
- b. **0.3** for obtaining the lake's name, area and perimeter from the user and converting the area and perimeter to a numeric type (0.2 input, 0.1 conversion)
- c. **0.3** for creating a lake object using the name, area and perimeter specified by the user
- d. **0.2** for invoking the SDIndex method for the lake object
- e. **0.2** for displaying the lake's name and the value returned by the SDIndex method
- f. **0.2** for checking the value of the shoreline development index to determine whether the lake is round or not, using an if statement
- g. **0.1** for displaying an appropriate message indicating whether the lake is round or not
- h. **0.6** for the screen captures requested showing the code and results