PSP Final Practical Exam Practise questions:

Python

- Write a Python application that reads a series of azimuths from a sequential file and converts the azimuths to bearings (in decimal degrees). The application displays the azimuths and their corresponding bearings in columns with headers at the top of the columns as shown. Use exceptions to validate file access, the azimuth values read from the file and include a general exception for unanticipated system errors. Convert the azimuth in dd to a bearing in degrees minutes seconds

Azimuth(dd)	**Bearing (dd)**	**Bearing (dms)**
45.500000	N 45.500000 E	N 45d 30' 00" E
100.200000	S 79.800000 E	S 79d 48' 00" E
214.560000	S 34.560000 W	S 34d 33' 36" W
303.987654	N 56.012346 W	N 56d 00' 44.4456" W

To convert from azimuths to bearings or bearings to azimuths:

- Determine the proper quadrant letters using:
  0° – 90° is NE
  90° – 180° is SE
  180° – 270° is SW
  270° – 360° is NW
- Determine the numerical value using:
  NE quadrant: bearing = azimuth
  SE quadrant: bearing = 180° – azimuth or azimuth = 180 - bearing
  SW quadrant: bearing = azimuth – 180° or azimuth = bearing + 180
  NW quadrant: bearing = 360° - azimuth or azimuth = 360 - bearing

- Write a Python application that uses a programmer-defined Python Class to represent an angle as a bearing. The class should have public attributes for degrees, minutes, seconds and the directon of the angle (N, S, E, W). The class should also have a public method for converting the angle to an azimuth in decimal degrees. The main application should allow a user to enter a series of bearings using the keyboard and get their azimuths using the programmer defined class. The main application should write the bearings and their corresponding azimuths to a sequential file.

Bearing (dms)	**Bearing (dd)**	**Azimuth(dd)**
N 45d 30' 00" E	N 45.500000 E	45.500000
S 79d 48' 00" E	S 79.800000 E	100.200000
S 34d 33' 36" W	S 34.560000 W	214.560000
N 56d 00' 44.4456" W	N 56.012346 W	303.987654

- Write a Python application that cleans up a string with repeating values. The application allows a user to enter in a comma delimited string which may contain values that repeat sequentially. If the string entered is not empty, the application calls a programmer-defined function to remove the sequentially repeating values from the string. Values in a string that repeat but are not sequential are not removed. The function returns the cleaned up string and the application displays the cleaned up string.

Input string to be checked	**String with No sequential repeats**
24,24,27,23,55,98,98,61,23,23,24,24	24,27,23,55,98,61,23,24

Field Calculations

In ArcGIS Pro's Calculate Field, create a programmer-defined function and call the function passing in fields or values as parameters.

– Perform a python mathematical operation for each record using the values in several fields (eg. max, min, sum, pow, sqrt, round, trig, degrees, radians, hypot, abs).
– Use string functions to rearrange the values from a field or combine the values from several fields

- Given a table with an Area field and a Population field, write a Python field calculate field script that determines the population density class based on the ratio of population to area for 4 different class ranges (eg. > 50 is class 1, 30-50 is class 2, 15-30 is class 3, 0-15 is class 4)
- Given a table with an address field consisting of number name type and direction (eg. 195 Lindsay St N), write a Python calculate field script that will replace the short form street type with its full word (eg. Street instead of St)

Geoprocessing (references provided below)

- Create a Python script that selects a subset of the features in a Feature Class using the Select tool and saves the selected features into a new feature class. Assume that the select query will be hard coded into the script. The script should then use the Near tool with the selected features and a point feature class to find the distance to the nearest selected feature from each point. The script will be attached to a Script tool in ArcGIS Pro. The script must get from parameters 1) the final feature class location and name and 2) the point feature class.
- Create a Python script that puts multiple polygon feature classes in a workspace together into a single feature class (using Union/Merge/Append tool). The script should then create a file geodatabase and bring the resulting "merged" feature class into the file geodatabase. The script will be attached to a Script tool in ArcGIS Pro. The script must get from parameters 1) the workspace containing the feature classes to be merged, 2) the name of the geodatabase and 3) the name of the new feature class

Map Scripting

- For an ArcGISProject file and a layer file (.lyr) given by the user, create a Python script that adds the layer file to the map, based on its shape type. If the layer contains points, it should be added at the top of the map, if the layer contains lines, it should be added either before the lines or after the lines, if the layer contains polygons, it should be added after the polygons. Hint, take a look at insertlayer, moveLayer and addLayer, do not use "AUTO_ARRANGE"
- Create a Python script that asks the user for the name of a folder and the name of the user's organization. The script then finds all the project files in the folder. For each project file, the script should find the Layout element for the credits and change the credits to contain today's date (using a Python built in function) and the name of the organization. In addition, the script should place the credits at the bottom of the page, centered under the map frame.

Map Scripting and Geoprocessing (reference provided below)

- Create a Python script that asks a user for the name of a layer (feature class), the name of a project file, the type of symbology desired (unique values, graduated colour, graduated symbol), the name of the field to symbolize and the number of classes. Assume a separate predefined layer file (.lyr) already exists for each type of symbology and the location of the predefined layer files is known by the programmer. The script should search in the project file for the layer specified by the user, then symbolize the layer with one of the three predefined layer files (.lyr) based on the type of symbology desired by the user (using the Apply Symbology From Layer tool). Assume the predefined layer file uses a field that already exists in the feature class. After the symbology has been applied the script should then set the field to use for symbology and number of classes.

References

Select Tool

Syntax:

Select_analysis (in_features, out_feature_class, {where_clause})

Parameter	Explanation	Data Type
in_features	The input feature class or layer from which features are selected.	Feature Layer
out_feature_class	The output feature class to be created. If no expression is used, it contains all input features.	Feature Class
where_clause (Optional)	An SQL expression used to select a subset of features. For more information on SQL syntax see the help topic	SQL Expression

Select Example (stand-alone Python Script)

# Name: Select_Example2.py

```
# Description: Select roads of Class 4 from major roads tin the gnatcatcher habitat study area
# Import system modules
import arcpy
from arcpy import env

# Set workspace
env.workspace = "C:/data"

# Set local variables
in_features = "majorrds.shp"
out_feature_class = "C:/output/majorrdsClass4.shp"
where_clause = '"CLASS" = \'4\''

# Execute Select
arcpy.Select_analysis(in_features, out_feature_class, where_clause)
```

Near Tool

Syntax:

Near_analysis (in_features, near_features, {search_radius}, {location}, {angle}, {method}, {field_names})

Parameter	**Explanation**	**Data Type**
in_features	The input features that can be point, polyline, polygon, or multipoint type.	Feature Layer
near_features [near_features,...]	One or more feature layers or feature classes containing near feature candidates. The near features can be of point, polyline, polygon, or multipoint. If multiple layers or feature classes are specified, a field named NEAR_FC is added to the input table and will store the paths of the source feature class containing the nearest feature found. The same feature class or layer may be used as both input and near features.	Feature Layer
search_radius (Optional)	The radius used to search for near features. If no value is specified, all near features are considered. If a distance but no unit or unknown is specified, the units of the coordinate system of the input features are used. If the **Geodesic** option is used, a linear unit such as Kilometers or Miles should be used.	Linear unit
location (Optional)	Specifies whether x- and y-coordinates of the closest location of the near feature will be written to the NEAR_X and NEAR_Y fields. • NO_LOCATION — Location information will not be written to the output table. This is the default. • LOCATION — Location information will be written to the output table.	Boolean

angle (Optional)	Specifies whether the near angle will be calculated and written to a NEAR_ANGLE field in the output table. A near angle measures direction of the line connecting an input feature to its nearest feature at their closest locations. When the PLANAR method is used in the method parameter, the angle is within the range of -180° to 180°, with 0° to the east, 90° to the north, 180° (or -180°) to the west, and -90° to the south. When the GEODESIC method is used, the angle is within the range of -180° to 180°, with 0° to the north, 90° to the east, 180° (or -180°) to the south, and -90° to the west. • NO_ANGLE —The near angle values will not be written. This is the default. • ANGLE —The near angle values will be written to the NEAR_ANGLE field.	Boolean
method (Optional)	Determines whether to use a shortest path on a spheroid (geodesic) or a flat earth (planar) method. It is strongly suggested to use the **Geodesic** method with data stored in a coordinate system that is not appropriate for distance measurements (for example, Web Mercator or any geographic coordinate system) and any analysis that spans a large geographic area. • PLANAR —Uses planar distances between the features. This is the default. • GEODESIC —Uses geodesic distances between features. This method takes into account the curvature of the spheroid and correctly deals with data near the dateline and poles.	String
field_names [[property, fieldname],...] (Optional)	Specifies the names of the attribute fields that will be added during processing. If this parameter is not used or any fields that will be added are excluded from this parameter, the default field names will be used. By default, NEAR_FID and NEAR_DIST fields will always be added, NEAR_X and NEAR_Y fields will be added when the Location parameter (location in Python) is enabled, the NEAR_ANGLE field will be added when the Angle parameter (angle in Python) is enabled, and the NEAR_FC field will be added when multiple inputs are used.	Value Table

Near example (stand-alone Python script)

```
# Name: Near.py
# Description: Finds nearest features from input feature class to near feature class.
import arcpy
# Set workspace environment
arcpy.env.workspace = "C:/data/city.gdb"

# set local variables
in_features = "houses"
near_features = "parks"

# find features only within search radius
search_radius = "5000 Meters"

# find location nearest features
location = "LOCATION"

# avoid getting angle of nearest features
angle = "NO_ANGLE"
# execute the function
arcpy.Near_analysis(in_features, near_features, search_radius, location, angle)
```

Union Tool

Syntax:

Union_analysis (in_features, out_feature_class, {join_attributes}, {cluster_tolerance}, {gaps})

Parameter	Explanation	Data Type
in_features [[in_features, {Rank}],...]	A list of the input feature classes or layers. When the distance between features is less than the cluster tolerance, the features with the lower rank will snap to the feature with the higher rank. The highest rank is one. All of the input features must be polygons.	Value Table
out_feature_class	The feature class that will contain the results.	Feature Class
join_attributes (Optional)	Determines which attributes from the input features will be transferred to the output feature class. • ALL —All the attributes from the input features will be transferred to the output feature class. This is the default. • NO_FID —All the attributes except the FID from the input features will be transferred to the output feature class. • ONLY_FID —Only the FID field from the input features will be transferred to the output feature class.	String
cluster_tolerance (Optional)	The minimum distance separating all feature coordinates (nodes and vertices) as well as the distance a coordinate can move in X or Y (or both).	Linear unit

gaps (Optional)	Gaps are areas in the output feature class that are completely enclosed by other polygons. This is not invalid, but it may be desirable to identify these for analysis. To find the gaps in the output, set this option to NO_GAPS, and a feature will be created in these areas. To select these features, query the output feature class based on all the input feature's FID values being equal to -1. • GAPS —No feature will be created for areas in the output that are completely enclosed by polygons. This is the default. • NO_GAPS —A feature will be created for the areas in the output that are completely enclosed by polygons. This feature will have blank attributes and its FID values will be -1.	Boolean

Union Example (Stand-alone Script)

```
# unions.py
# Purpose: union 3 feature classes
# Import the system modules
import arcpy
from arcpy import env

# Set the current workspace
# (to avoid having to specify the full path to the feature classes each time)
env.workspace = "c:/data/data.gdb"

# Union 3 feature classes but only carry the FID attributes to the output
inFeatures = ["well_buff50", "stream_buff200", "waterbody_buff500"]
outFeatures = "water_buffers"
clusterTol = 0.0003
arcpy.Union_analysis (inFeatures, outFeatures, "ONLY_FID", clusterTol)

# Union 3 other feature classes, but specify some ranks for each
# since parcels has better spatial accuracy
inFeatures = [["counties", 2],["parcels", 1],["state", 2]]
outFeatures = "state_landinfo"
arcpy.Union_analysis (inFeatures, outFeatures)
```

Merge Tool

Syntax:

Merge_management (inputs, output, {field_mappings}, {add_source})

Parameter	Explanation	Data Type
inputs [inputs,...]	The input datasets that will be merged together into a new output dataset. Input datasets can be point, line, or polygon feature classes or tables. The input datasets must all be of the same type.	Table View

output	The output dataset that will contain all combined input datasets.	Feature Class; Table
field_mappings (Optional)	Controls how the attribute fields from the input datasets are mapped and transferred to the output dataset. You can add, rename, or delete output fields as well as set properties, such as data type and merge rule. Merge rules allow you to specify how values from two or more input fields are merged or combined into a single output value. There are several merge rules that determine how the output field is populated with values. • First—Use the input fields' first value. • Last—Use the input fields' last value. • Join—Concatenate (join) the input fields' values. • Sum—Calculate the total of the input fields' values. • Mean—Calculate the mean (average) of the input fields' values. • Median—Calculate the median (middle) of the input fields' values. • Mode—Use the value with the highest frequency. • Min—Use the minimum value of all input fields' values. • Max—Use the maximum value of all input fields' values. • Standard deviation—Use the standard deviation classification method on all input fields' values. • Count—Find the number of records included in the calculation.	Field Mappings
dd_source (Optional)	Specifies whether source information will be added to the output dataset in a new text field, MERGE_SRC. The values in the MERGE_SRC field will indicate the input dataset path or layer name that is the source of each record in the output. • NO_SOURCE_INFO — Source information will not be added to the output dataset in a MERGE_SRC field. This is the default. ADD_SOURCE_INFO —Source information will be added to the output dataset in a MERGE_SRC field.	Boolean

Merge example (stand-alone script)

```
# Name: Merge.py
# Description: Use Merge tool to move features from two street
#                  feature classes into a single dataset with field mapping
# import system modules
import arcpy
from arcpy import env
```

```
# Set environment settings
env.workspace = "C:/data"

# Street feature classes to be merged
oldStreets = "majorrds.shp"
newStreets = "Habitat_Analysis.gdb/futrds"
addSourceInfo = "ADD_SOURCE_INFO"

# Create FieldMappings object to manage merge output fields
fieldMappings = arcpy.FieldMappings()

# Add all fields from both oldStreets and newStreets
fieldMappings.addTable(oldStreets)
fieldMappings.addTable(newStreets)

# Add input fields "STREET_NAM" & "NM" into new output field
fldMap_streetName = arcpy.FieldMap()
fldMap_streetName.addInputField(oldStreets,"STREET_NAM")
fldMap_streetName.addInputField(newStreets,"NM")

# Set name of new output field "Street_Name"
streetName = fldMap_streetName.outputField
streetName.name = "Street_Name"
fldMap_streetName.outputField = streetName

# Add output field to field mappings object
fieldMappings.addFieldMap(fldMap_streetName)

# Add input fields "CLASS" & "IFC" into new output field
fldMap_streetClass = arcpy.FieldMap()
fldMap_streetClass.addInputField(oldStreets,"CLASS")
fldMap_streetClass.addInputField(newStreets,"IFC")

# Set name of new output field "Street_Class"
streetClass = fldMap_streetClass.outputField
streetClass.name = "Street_Class"
fldMap_streetClass.outputField = streetClass

# Add output field to field mappings object
fieldMappings.addFieldMap(fldMap_streetClass)

# Remove all output fields from the field mappings, except fields "Street_Class",
"Street_Name", & "Distance"
for field in fieldMappings.fields:
   if field.name not in ["Street_Class","Street_Name","Distance"]:
      fieldMappings.removeFieldMap(fieldMappings.findFieldMapIndex(field.name))

# Since both oldStreets and newStreets have field "Distance", no field mapping is required
# Use Merge tool to move features into single dataset
uptodateStreets = "C:/output/Output.gdb/allroads"
arcpy.Merge_management([oldStreets, newStreets], uptodateStreets, fieldMappings)
```

Append Tool

Syntax:

Append_management (inputs, target, {schema_type}, {field_mapping}, {subtype}, {expression})

Parameter	Explanation	Data Type
inputs [inputs,...]	The input datasets whose data will be appended into the target dataset. Input datasets can be point, line, or polygon feature classes, tables, rasters, raster catalogs, annotation feature classes, or dimensions feature classes. Each input dataset must match the data type of the target dataset.	Table View; Raster Layer
target	The existing dataset that the input datasets' data will be appended into. Each input dataset must match the data type of the target dataset.	Table View; Raster Layer
schema_type (Optional)	Specifies if the schema (field definitions) of the input datasets must match the schema of the target dataset in order for data to be appended. • TEST —Input dataset schema (field definitions) must match the schema of the target dataset. An error will be returned if the schemas do not match. • NO_TEST —Input dataset schema (field definitions) do not have to match that of the target dataset. Any fields from the input datasets that do not match the fields of the target dataset will not be mapped to the target dataset unless the mapping is explicitly set in the Field Map control.	String
field_mapping (Optional)	Controls how the attribute information in input datasets' fields is transferred to the target dataset. This parameter can only be used if the **Schema Type** NO_TEST is specified. Because the input datasets' data is appended into an existing target dataset that has a predefined schema (field definitions), fields cannot be added or removed from the target dataset. Merge rules allow you to specify how values from two or more input fields are merged or combined into a single output value. There are several merge rules that determine how the output field is populated with values. • First—Use the input fields' first value. • Last—Use the input fields' last value. • Join—Concatenate (join) the input fields' values. • Sum—Calculate the total of the input fields' values. • Mean—Calculate the mean (average) of the input fields' values. • Median—Calculate the median (middle) of the input	Field Mapping

	fields' values. • Mode—Use the value with the highest frequency. • Min—Use the minimum value of all input fields' values. • Max—Use the maximum value of all input fields' values. • Standard deviation—Use the standard deviation classification method on all input fields' values. • Count—Find the number of records included in the calculation.	
subtype (Optional)	A subtype description to assign that subtype to all new data that is appended to the target dataset.	String
expression (Optional)	The SQL expression used to select a subset of the input datasets' records. If multiple input datasets are specified, they will all be evaluated against the expression. If no records match the expression for an input dataset, no records from that dataset will be appended to the target.	SQL Expression

Append example (stand-alone Python script)

```
# Name: Append.py
# Description: Use the Append tool to combine several shapefiles
# import system modules
import arcpy
import os

# Set environment settings
arcpy.env.workspace = "C:/data"

# Set local variables
outLocation = "C:/Output"
emptyFC = "MA_towns.shp"
schemaType = "NO_TEST"
fieldMappings = ""
subtype = ""

# Process:  Create a new empty feature class to append shapefiles into
arcpy.CreateFeatureclass_management(outLocation, emptyFC, "POLYGON", "amherst.shp")

# All polygon FCs in the workspace are MA town shapefiles, we want to append these to the
empty FC
fcList = arcpy.ListFeatureClasses("","POLYGON")

# list will resemble ["amherst.shp", "hadley.shp", "pelham.shp", "coldspring.shp"]
# Process: Append the feature classes into the empty feature class
arcpy.Append_management(fcList, outLocation + os.sep + emptyFC, schemaType,
fieldMappings, subtype)
```

CreateFileGDB Tool

Syntax:

CreateFileGDB_management (out_folder_path, out_name, {out_version})

Parameter	Explanation	Data Type
out_folder_path	Location (folder) where the file geodatabase will be created.	Folder
out_name	Name of the geodatabase to be created.	String
out_version (Optional)	The ArcGIS version for the geodatabase to be created. • CURRENT —Creates a geodatabase compatible with the currently installed version of ArcGIS • 10.0 —Creates a geodatabase compatible with ArcGIS version 10 • 9.3 —Creates a geodatabase compatible with ArcGIS version 9.3 • 9.2 —Creates a geodatabase compatible with ArcGIS version 9.2	String

CreateFileGDB Example (Stand-alone Python Script)

```
# Name: CreateFileGDB_Example2.py
# Import system modules
import arcpy
from arcpy import env

# Set local variables
out_folder_path = "C:/output"
out_name = "fGDB.mdb"

# Execute CreateFileGDB
arcpy.CreateFileGDB_management(out_folder_path, out_name)
```

FeatureClassToFeatureClass Tool

Syntax:

FeatureClassToFeatureClass_conversion (in_features, out_path, out_name, {where_clause}, {field_mapping}, {config_keyword})

Parameter	Explanation	Data Type
in_features	The feature class or feature layer that will be converted.	Feature Layer
out_path	The location in which the output feature class will be created. This can be either a geodatabase or a folder. If the output location is a folder, the output will be a shapefile.	Workspace; Feature Dataset
out_name	The name of the output feature class.	String
where_clause (Optional)	An SQL expression used to select a subset of features.	SQL Expression
field_mapping (Optional)	The fields and field contents chosen from the input feature class. You can add, rename, or delete output fields as well	Field Mappings

	as set properties such as data type and merge rule. First—Use the input fields' first value. Last—Use the input fields' last value. Join—Concatenate (join) the input field values. Sum—Calculate the total of the input field values. Mean—Calculate the mean (average) of the input field values. Median—Calculate the median (middle) of the input field values. Mode—Use the value with the highest frequency. Min—Use the minimum value of all the input field values. Max—Use the maximum value of all the input field values. Standard deviation—Use the standard deviation classification method on all the input field values. Count—Find the number of records included in the calculation.	
config_keyword (Optional)	Specifies the storage parameters (configuration) for geodatabases in file and enterprise geodatabases. Personal geodatabases do not use configuration keywords.	String

FeatureClassToFeatureClass example 2 (stand-alone script)

```
# Name: FeatureClassToFeatureClass_Example2.py
# Description: Use FeatureClassToFeatureClass with an expression to create a subset
#  of the original feature class.
# Import system modules
import arcpy

# Set environment settings
arcpy.env.workspace = "C:/data/GreenvalleyDB.mdb/Public Buildings"

# Set local variables
inFeatures = "buildings_point"
outLocation = "C:/output/output.gdb"
outFeatureClass = "postoffices"
delimitedField = arcpy.AddFieldDelimiters(env.workspace, "NAME")
expression = delimitedField + " = 'Post Office'"

# Execute FeatureClassToFeatureClass
arcpy.FeatureClassToFeatureClass_conversion(inFeatures, outLocation,
                            outFeatureClass, expression)
```

ApplySymbologyFromLayer Tool

Syntax:

ApplySymbologyFromLayer(in_layer, in_symbology_layer, {symbology_fields}, {update_symbology})

Parameter	Explanation	Data Type
in_layer	The layer to which the symbology will be applied.	Feature Layer; Raster Layer; Layer
in_symbology_layer	The symbology of this layer will be applied to the input layer. Both .lyrx and .lyr files are supported.	Layer
symbology_fields [[field_type, source_field, target_field],...] (Optional)	The fields from the input layer that match the symbology fields used in the symbology layer. Symbology fields contain three properties: • Field type—Specifies the field type: symbology value, normalization, or other type. • Source field—The symbology field used by the symbology layer. Use a blank value or "#" if you do not know the source field and want to use the default. • Target field—The field from the input layer to use when applying the symbology. Supported field types are as follows: • VALUE_FIELD—Primary field used to symbolize values • NORMALIZATION_FIELD—Field used to normalize quantitative values • EXCLUSION_CLAUSE_FIELD—Field used for the symbology exclusion clause. • CHART_RENDERER_PIE_SIZE_FIELD—Field used to set the size of pie chart symbols • ROTATION_XEXPRESSION_FIELD—Field used to set the rotation of symbols on the x-axis • ROTATION_YEXPRESSION_FIELD—Field used to set the rotation of symbols on the y-axis • ROTATION_ZEXPRESSION_FIELD—Field used to set the rotation of symbols on the z-axis • TRANSPARENCY_EXPRESSION_FIELD—Field used to set the transparency of symbols • TRANSPARENCY_NORMALIZATION_FIELD—Field used to normalize transparency values • SIZE_EXPRESSION_FIELD—Field used to set the size or width of symbols • COLOR_EXPRESSION_FIELD—Field used to set the color of symbols • PRIMITIVE_OVERRIDE_EXPRESSION_FIELD—Field used to set various properties on individual symbol layers	Value Table

update_symbology (Optional)	Specifies whether symbology ranges will be updated. • DEFAULT —Symbology ranges will be updated, except in the following situations: ○ When the input layer is empty ○ When the symbology layer uses class breaks (for example, graduated colors or graduated symbols) and the classification method is manual or defined interval ○ When the symbology layer uses unique values and the Show all other values option is checked • UPDATE —Symbology ranges will be updated. • MAINTAIN —Symbology ranges will not be updated; they will be maintained.	String

ApplySymbologyFromLayer example 2 (stand-alone script)

```
# Import system modules
import arcpy

# Set the current workspace
arcpy.env.workspace = "C:/data.gdb"

# Set layer to apply symbology to
inputLayer = "sf_points"

# Set layer that output symbology will be based on
symbologyLayer = "water_symbols_pnt.lyrx"

# Apply the symbology from the symbology layer to the input layer
arcpy.ApplySymbologyFromLayer_management(inputLayer, symbologyLayer)
```