Exercise 12
Creating Python functions and classes

Create Python functions

In this exercise, you will create a custom function that can be called from within the same script or from another script.

1 Start PythonWin. Create a new Python script and save as list.py **to the C:\EsriPress\Python\Data\Exercise12\Results folder.**

2 Enter the following code:

```
import arcpy
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = arcpy.ListFields("streets.shp")
print fields
```

Running this code creates a list of fields.

3 Save and run the script.

Running the script prints the Python reference information for each field object in the streets shapefile. The code does not print any field names. The output prints as follows:

```
  [<Field object at 0xfb9870[0xe021e90]>, <Field object at ➜
➜ 0xe0d33f0[0xe021f38]>, <Field object at 0xe0d3570[0xe021f68]>, ➜
➜ <Field object at 0xe0d37b0[0xe104038]>, <Field object at ➜
➜ 0xe0d3b90[0xe1040b0]>,...
```

To print the names of the field objects, you can use their `name` property. A `for` loop can be used to iterate over the list of fields.

4 Modify the code as follows:

```
import arcpy
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = arcpy.ListFields("streets.shp")
namelist = []
for field in fields:
    namelist.append(field.name)
print namelist
```

5 Save and run the script.

Running the script prints the field names, as follows:

```
[u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_ ➔
➔ NAM', u'STREET_TYP', u'SUF_DIR', u'FULLNAME', ...]
```

Once you have the script to create a list of field names, you may want to use it again. You can do this by creating a custom function, which you'll do next.

6 Modify the code as follows:

```
import arcpy

def listfieldnames(table):
    fields = arcpy.ListFields(table)
    namelist = []
    for field in fields:
        namelist.append(field.name)
    return namelist
```

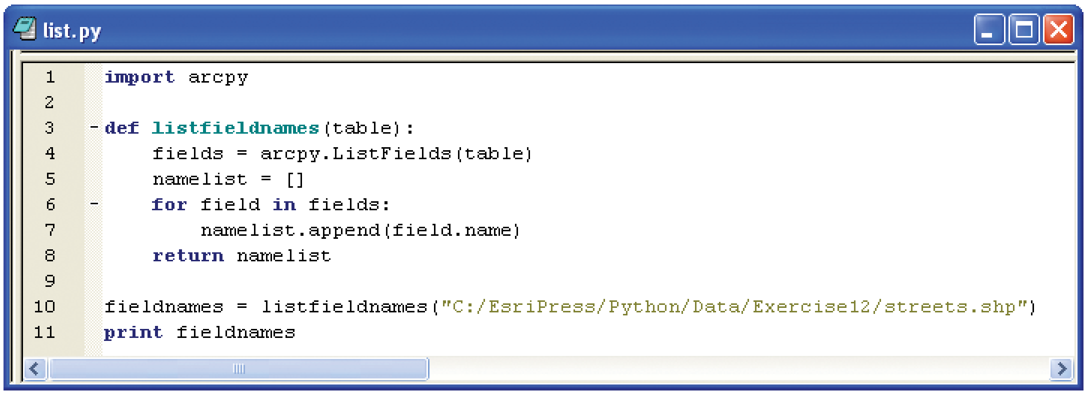
It is common to skip a line in a script before a custom function. This has no effect on running the script.

The block of code that creates the list of names is now defined as a function called `listfieldnames`. This function can now be called, for example, from within the same script. You'll do this next.

7 Add the following lines of code:

```
    fieldnames = listfieldnames("C:/EsriPress/Python/Data/Exercise12/ ➔
➔   streets.shp")
    print fieldnames
```

The complete script now looks like the example in the figure.


list.py

```
import arcpy

def listfieldnames(table):
    fields = arcpy.ListFields(table)
    namelist = []
    for field in fields:
        namelist.append(field.name)
    return namelist

fieldnames = listfieldnames("C:/EsriPress/Python/Data/Exercise12/streets.shp")
print fieldnames
```

8 Save and run the script. Running the script prints the list of field names to the Interactive Window.

Initially, it does not appear to do anything different from the earlier version of the script that did not define a custom function. However, once the function is created, it can also be called from another script. You will do that next.

Call functions from other scripts

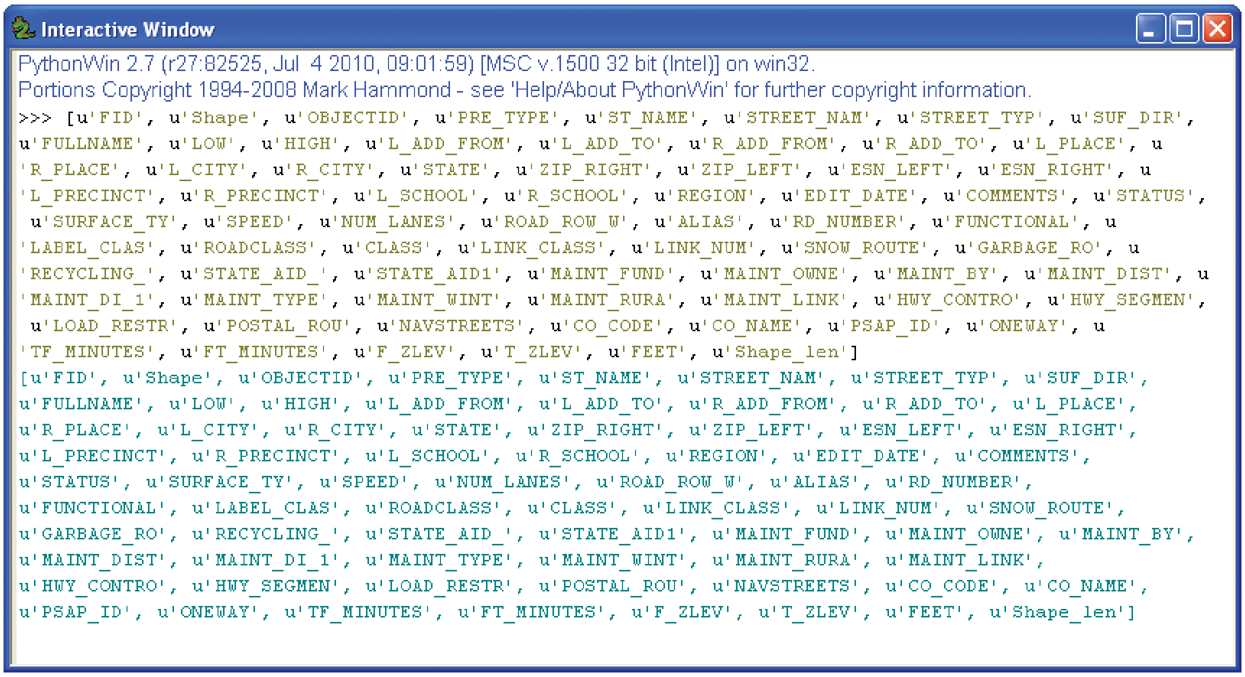
1 In PythonWin, create a new Python script and save as myscript.py **to the Results folder for exercise 12.**

2 Enter the following code:

```
import arcpy
import list
arcpy.env.workspace = "C:/EsriPress/Python/Data/Exercise12"
fields = list.listfieldnames("streets.shp")
print fields
```

Running this script imports the `list` module, calls the `listfieldnames` function, and passes the name of a table as an argument to this function.

3 Save and run the script. The result is that the list of field names is printed twice to the Interactive Window.



```
PythonWin 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> [u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_NAM', u'STREET_TYP', u'SUF_DIR',
u'FULLNAME', u'LOW', u'HIGH', u'L_ADD_FROM', u'L_ADD_TO', u'R_ADD_FROM', u'R_ADD_TO', u'L_PLACE', u
'R_PLACE', u'L_CITY', u'R_CITY', u'STATE', u'ZIP_RIGHT', u'ZIP_LEFT', u'ESN_LEFT', u'ESN_RIGHT', u
'L_PRECINCT', u'R_PRECINCT', u'L_SCHOOL', u'R_SCHOOL', u'REGION', u'EDIT_DATE', u'COMMENTS', u'STATUS',
 u'SURFACE_TY', u'SPEED', u'NUM_LANES', u'ROAD_ROW_W', u'ALIAS', u'RD_NUMBER', u'FUNCTIONAL', u
'LABEL_CLAS', u'ROADCLASS', u'CLASS', u'LINK_CLASS', u'LINK_NUM', u'SNOW_ROUTE', u'GARBAGE_RO', u
'RECYCLING_', u'STATE_AID_', u'STATE_AID1', u'MAINT_FUND', u'MAINT_OWNE', u'MAINT_BY', u'MAINT_DIST', u
'MAINT_DI_1', u'MAINT_TYPE', u'MAINT_WINT', u'MAINT_RURA', u'MAINT_LINK', u'HWY_CONTRO', u'HWY_SEGMEN',
 u'LOAD_RESTR', u'POSTAL_ROU', u'NAVSTREETS', u'CO_CODE', u'CO_NAME', u'PSAP_ID', u'ONEWAY', u
'TF_MINUTES', u'FT_MINUTES', u'F_ZLEV', u'T_ZLEV', u'FEET', u'Shape_len']
[u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_NAM', u'STREET_TYP', u'SUF_DIR',
u'FULLNAME', u'LOW', u'HIGH', u'L_ADD_FROM', u'L_ADD_TO', u'R_ADD_FROM', u'R_ADD_TO', u'L_PLACE',
u'R_PLACE', u'L_CITY', u'R_CITY', u'STATE', u'ZIP_RIGHT', u'ZIP_LEFT', u'ESN_LEFT', u'ESN_RIGHT',
u'L_PRECINCT', u'R_PRECINCT', u'L_SCHOOL', u'R_SCHOOL', u'REGION', u'EDIT_DATE', u'COMMENTS',
u'STATUS', u'SURFACE_TY', u'SPEED', u'NUM_LANES', u'ROAD_ROW_W', u'ALIAS', u'RD_NUMBER',
u'FUNCTIONAL', u'LABEL_CLAS', u'ROADCLASS', u'CLASS', u'LINK_CLASS', u'LINK_NUM', u'SNOW_ROUTE',
u'GARBAGE_RO', u'RECYCLING_', u'STATE_AID_', u'STATE_AID1', u'MAINT_FUND', u'MAINT_OWNE', u'MAINT_BY',
u'MAINT_DIST', u'MAINT_DI_1', u'MAINT_TYPE', u'MAINT_WINT', u'MAINT_RURA', u'MAINT_LINK',
u'HWY_CONTRO', u'HWY_SEGMEN', u'LOAD_RESTR', u'POSTAL_ROU', u'NAVSTREETS', u'CO_CODE', u'CO_NAME',
u'PSAP_ID', u'ONEWAY', u'TF_MINUTES', u'FT_MINUTES', u'F_ZLEV', u'T_ZLEV', u'FEET', u'Shape_len']
```

What happened? When the myscript.py script called the `listfieldnames` function, it ran the list.py script. This script creates and then prints the list of field names. The function also returns the list of field names, and it is printed by the myscript.py script.

To avoid double printing, some additional code is needed, which you'll add next.

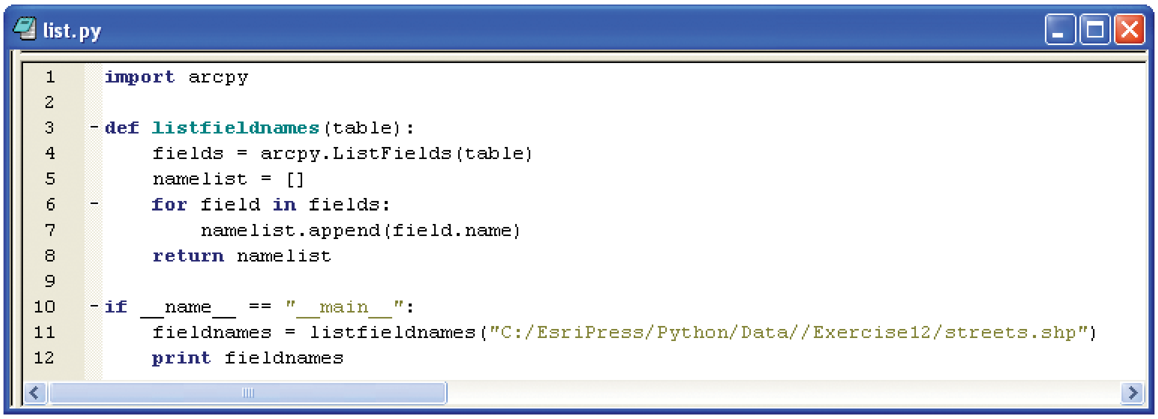
Note: *If PythonWin had been left open with the list.py script open from the previous steps, the script would not have been loaded again, and the result would have printed only once.*

4 Open the list.py script and add the following line of code just before the line that starts with " `fieldnames ...` **":**

```
    if __name__ == "__main__":
```

Note: *There are two underscores in each spot.*

5 Indent the last two lines of code. The script should now look like the example in the figure.


list.py

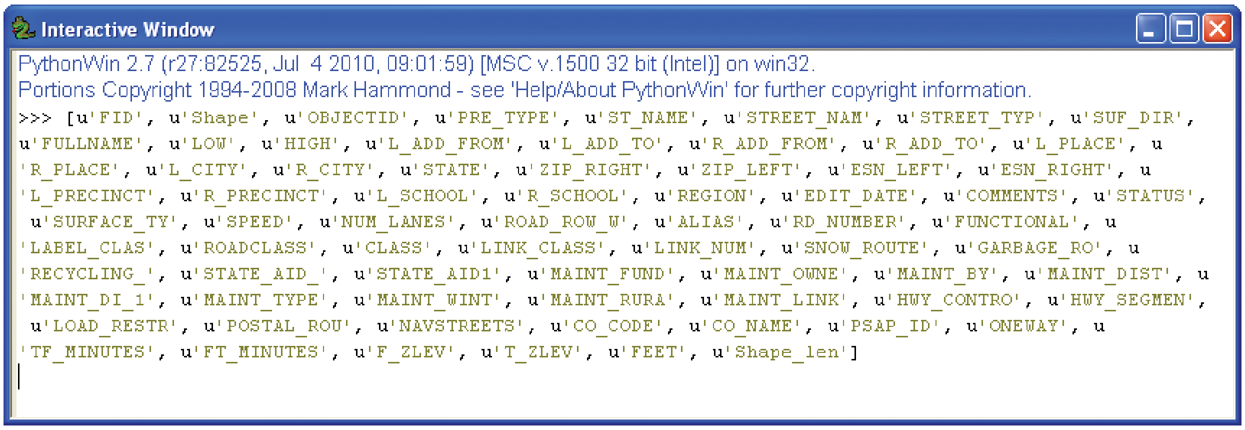
```
import arcpy

def listfieldnames(table):
    fields = arcpy.ListFields(table)
    namelist = []
    for field in fields:
        namelist.append(field.name)
    return namelist

if __name__ == "__main__":
    fieldnames = listfieldnames("C:/EsriPress/Python/Data//Exercise12/streets.shp")
    print fieldnames
```

6 Save the list.py script.

7 Close PythonWin.

8 Start PythonWin and open the myscript.py script.

9 Without making any changes, run the script. Running the script prints the list of field names only once to the Interactive Window.

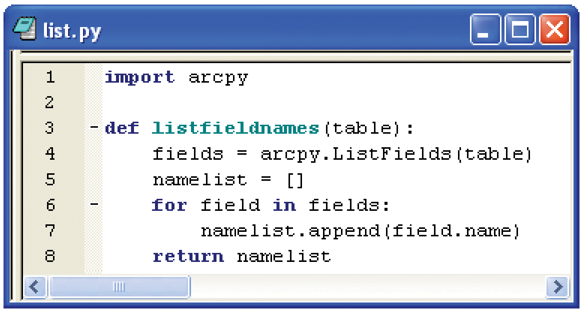

Interactive Window

```
PythonWin 2.7 (r27:82525, Jul  4 2010, 09:01:59) [MSC v.1500 32 bit (Intel)] on win32.
Portions Copyright 1994-2008 Mark Hammond - see 'Help/About PythonWin' for further copyright information.
>>> [u'FID', u'Shape', u'OBJECTID', u'PRE_TYPE', u'ST_NAME', u'STREET_NAM', u'STREET_TYP', u'SUF_DIR',
u'FULLNAME', u'LOW', u'HIGH', u'L_ADD_FROM', u'L_ADD_TO', u'R_ADD_FROM', u'R_ADD_TO', u'L_PLACE', u
'R_PLACE', u'L_CITY', u'R_CITY', u'STATE', u'ZIP_RIGHT', u'ZIP_LEFT', u'ESN_LEFT', u'ESN_RIGHT', u
'L_PRECINCT', u'R_PRECINCT', u'L_SCHOOL', u'R_SCHOOL', u'REGION', u'EDIT_DATE', u'COMMENTS', u'STATUS',
 u'SURFACE_TY', u'SPEED', u'NUM_LANES', u'ROAD_ROW_W', u'ALIAS', u'RD_NUMBER', u'FUNCTIONAL', u
'LABEL_CLAS', u'ROADCLASS', u'CLASS', u'LINK_CLASS', u'LINK_NUM', u'SNOW_ROUTE', u'GARBAGE_RO', u
'RECYCLING_', u'STATE_AID_', u'STATE_AID1', u'MAINT_FUND', u'MAINT_OWNE', u'MAINT_BY', u'MAINT_DIST', u
'MAINT_DI_1', u'MAINT_TYPE', u'MAINT_WINT', u'MAINT_RURA', u'MAINT_LINK', u'HWY_CONTRO', u'HWY_SEGMEN',
 u'LOAD_RESTR', u'POSTAL_ROU', u'NAVSTREETS', u'CO_CODE', u'CO_NAME', u'PSAP_ID', u'ONEWAY', u
'TF_MINUTES', u'FT_MINUTES', u'F_ZLEV', u'T_ZLEV', u'FEET', u'Shape_len']
```

When the `myscript.py` script runs and calls the `listfieldnames` function, the `if __name__ == "__main__":` statement in the list.py script ensures that the next block of code is run only if the list.py script is run by itself.

The block of code following the `if __name__ == "__main__":` statement can be considered a "test." When the list.py script is run by itself, this test code allows you to check whether the custom function works correctly. However, it may not be necessary to keep this code if the script is being used to store a custom function that will only be called from other scripts.

10 **Modify the list.py script by removing the last three lines of code.** The script should now look like the example in the figure.



list.py

```
import arcpy

def listfieldnames(table):
    fields = arcpy.ListFields(table)
    namelist = []
    for field in fields:
        namelist.append(field.name)
    return namelist
```

11 **Save and close the list.py and myscript.py scripts.**

Work with classes

Classes allow you to group functions and variables together. Once created, classes make it possible to create objects that have specific properties as defined by these functions and variables.

Next, you will first consider a script to calculate property taxes, and then you will create a class to make this calculation more versatile.

1 **In PythonWin, create a new Python script and save as** assessment.py **to the Results folder for exercise 12.**

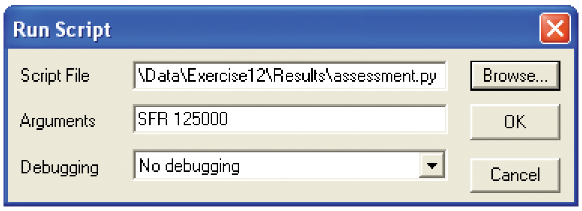
2 **Enter the following code:**

```
import sys
landuse = sys.argv[1]
value = int(sys.argv[2])

if landuse == "SFR":
    rate = 0.05
elif landuse == "MFR":
    rate = 0.04
else:
    rate = 0.02
assessment = value * rate
print assessment
```

This script calculates the property tax assessment based on variables for land use and the property value.

3 Save the script. The script requires two arguments.

4 Click the Run button. On the Run Script dialog box, enter the arguments as shown in the figure.



Script File: \Data\Exercise12\Results\assessment.py
Arguments: SFR 125000
Debugging: No debugging

5 Click OK. Running the script prints the result of `6250.0` to the Interactive Window.

To automate this calculation for many different entries, you would read the values from a file and iterate over these values. Then you would have a few options to carry out the tax calculation. First, you can place the code within the iteration—that is, within the `for` loop or the `while` loop. Second, you can create a custom function in a separate script that does the calculation; when the function is called, the necessary arguments are passed, and the function returns a value. Third, you can create a class that contains the calculation as a method.

Next, you will take a look at the use of a custom function.

6 In PythonWin, create a new Python script and save as tax.py **to the Results folder for exercise 12.**

7 Enter the following code:

```
def taxcalc(landuse, value):
    if landuse == "SFR":
        rate = 0.05
    elif landuse == "MFR":
        rate = 0.04
    else:
        rate = 0.02
    assessment = value * rate
    return assessment
```

8 Save your script.

9 Create a new Python script and save as parcelCalc.py **to the Results folder for exercise 12.**

10 Enter the following code:

```
import tax
mytax = tax.taxcalc("SFR", 125000)
print mytax
```

11 Save and run the script. Running the script prints the result of 6250.0 to the Interactive Window.

Next, you will look at the use of a class.

12 Create a new Python script and save as parcelclass.py **to the Results folder for exercise 12.**

13 Enter the following code:

```
class Parcel:
    def __init__(self, landuse, value):
        self.landuse = landuse
        self.value = value

    def assessment(self):
        if self.landuse == "SFR":
            rate = 0.05
        elif self.landuse == "MFR":
            rate = 0.04
        else:
            rate = 0.02
        assessment = self.value * rate
        return assessment
```

14 Save your script.

15 Create a new Python script and save as parcelTax.py **to the Results folder for exercise 12.**

16 Enter the following code:

```
import parcelclass
myparcel = parcelclass.Parcel("SFR", 125000)
print "Land use: ", myparcel.landuse
mytax = myparcel.assessment()
print "Tax assessment: ", mytax
```

17 Save and run the script.

Running the script prints the following tax information to the Interactive Window:

```
Land use: SFR
Tax assessment: 6250.0
```

Although the use of the class accomplishes the same calculation as the custom function, a class is more versatile because it allows you to combine properties and functions. It would be relatively easy, for example, to expand the class with additional methods for different calculations, which could all be part of the same class.

Challenge exercises

Challenge 1

Create a custom function called **countstringfields** that determines the number of fields of type `string` in an input feature class. Create this function in a separate script (for example, **mycount.py**) that you call from another script (for example, **callingscript.py**). You can use the streets.shp feature class in the Exercise12 folder.

Challenge 2

You are given a feature class called parcels.shp located in the Exercise12 folder that contains the following fields: FID, Shape, Landuse, and Value. Modify the parceltax.py script so that it determines the property tax for each parcel and stores these values in a list. You should use the class created in the parcelclass.py script—the class can remain unchanged. Print the values of the final list as follows:

```
FID: <property tax>
```