

Reading and Editing Cells

Working with Excel in Python

1

This video will discuss how to read, write, and format cells in an Excel worksheet.

The cell object

- Allows contents of cell(s) to be read or edited.

- Get cell object with cell name...

`worksheet.Range("A5")`

- Or get with cell's row, column...

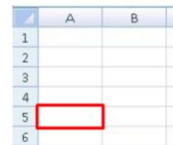
`worksheet.Cells(5,1)`

row ↗ ↘ column

- Get multiple cells...

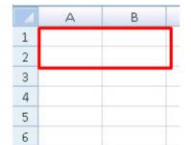
`worksheet.Range("A1:B2")`

`Range("A5")`
or `Cells(5,1)`



	A	B
1		
2		
3		
4		
5		
6		

`Range("A1:B2")`



	A	B
1		
2		
3		
4		
5		
6		

The cell object needs to be retrieved in order to read, edit, or format the contents of a cell. There are multiple ways to get the cell object for a single cell or a range of cells.

The worksheet object's **Range** method will retrieve a cell by its name.

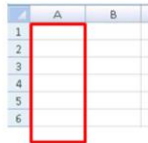
The **Cells** method will retrieve a cell by its row and column.

The **Range** method can retrieve a group of cells by specifying a starting cell and an ending cell separated by a colon. Note that the ending cell can be in a different column than the starting cell.

Column and row objects

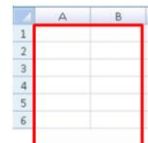
- Use for formatting columns/rows.
- Get column(s)...

worksheet.Columns("A")



	A	B
1		
2		
3		
4		
5		
6		

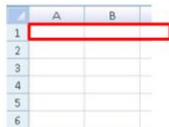
worksheet.Columns("A:B")



	A	B
1		
2		
3		
4		
5		
6		

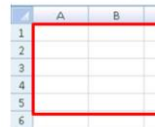
- Get row(s)...

worksheet.Rows("1")



	A	B
1		
2		
3		
4		
5		
6		

worksheet.Rows("1:5")



	A	B
1		
2		
3		
4		
5		
6		

3

Columns and rows can be retrieved for formatting purposes.

The **Columns** method can retrieve either a single column or a group of columns.

The **Rows** method can retrieve a single row or a group of rows.

Read/edit a cell

```
cell = worksheet.Range("A5")
```

- Read value in cell...

```
cell.Value
```

Note: reading from an empty cell gives a value of None

- Read formula in cell...

```
cell.Formula
```

Note: getting the cell object and reading/writing to a cell is typically combined into one statement...

- Write value to cell...

```
cell.Value = 3.14159
```

- Write formula to cell...

```
cell.Formula = "=PI()"
```

```
worksheet.Range("A5").Value
```

4

Once we've retrieved a cell object,

We can read the **Value** in the cell. The value will have the same data type as the cell in the spreadsheet. Note that reading a value from an empty cell will return a value of **None**.

We can read the **Formula** from the cell. The formula will be returned as a string – an empty cell will return an empty string.

The value of the cell can be set to any number or string value.

A formula can be set for a cell. The formula should be entered as a string; the contents of the string should be identical to a formula that would be typed directly in Excel. Note that formulas in Excel always start with an equal sign.

The steps of getting the cell object and accessing the value or formula properties are typically combined into a single statement.

Example: working with a spreadsheet

- Use a loop to work through a spreadsheet...

```
# get worksheet...
worksheet = workbook.Worksheets("Sheet1")

# for each row
for xlRow in xrange(1, 100000):
    # cell name...
    cell = "A%s" % xlRow
    # read cell value...
    value = worksheet.Range(cell).Value

    # break loop if at end of spreadsheet (value = None)
    if value == None: break
```

starting row number
max row number
substitute row number into cell name

This slide will show an example of how to iterate through the rows in a spreadsheet until the final row of data is processed. The first statement gets the worksheet object for "Sheet1".

A for loop is used to iterate through the rows in the sheet. Note that the **xrange** function is similar to the **range** function when used in a for loop. The functions differ in when they create the sequence of numbers: range creates the entire sequence and stores it in a list before the 1st iteration of the loop; however, xrange generates a single new value with each loop iteration. xrange is more efficient when used with for loops because it does not require the entire sequence to be stored in a list.

The first statement in the loop creates the cell name based on the loop iteration number. The next statement gets the value of the cell and assigns it to a variable.

The final statement tests the variable for a null value. If the value is None, then the loop will stop. Note that this script assumes there should be no blank cells, in column A, before the end of the spreadsheet.

Interactive scripts

- Excel can be used interactively with Python
 - changes made by script in Excel are immediately visible
 - changes made in Excel are immediately accessible by script
 - Note: Python will crash if trying to read cell currently being edited in Excel.
- Script should pause if waiting for input from Excel...

```
while 1:
    if worksheet.Range("A2").Value <> None:
        break
    time.sleep(5)
```

Annotations:

- create endless loop (points to `while 1:`)
- check if value has been entered in cell (points to `worksheet.Range("A2").Value <> None`)
- stop loop when value found in cell (points to `break`)
- pause script (5 seconds) (points to `time.sleep(5)`)

6

Python works interactively with Excel when the application is made visible (see 10a lecture video). Changes made through Python can be seen immediately in Excel and changes in Excel are immediately accessible by the script and without the need to save the spreadsheet. The interactivity is convenient when developing a script; however, the script will crash if it tries to work with a cell that is currently being edited in Excel.

A script can use an Excel spreadsheet as a user interface in which case, the script would need to periodically check the spreadsheet for information provided by the user.

Periodic checks could be made by setting up an endless while loop.

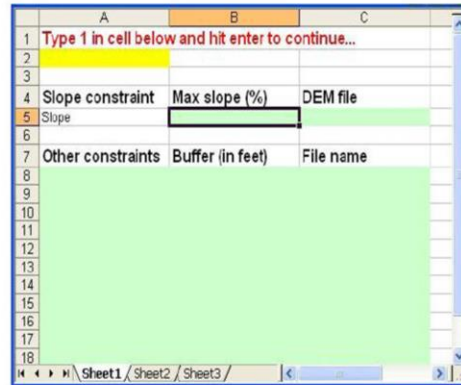
In the loop, the appropriate cell in the worksheet would be checked for a non-null value which would indicate that the user has finished specifying a value.

If the user-defined value has been received, then the loop will stop and the script can proceed.

If a user-defined value was not found, then the script will sleep for a period of time before continuing with the next iteration of the loop. Note that a short sleep time should be avoided because it would increase the chances that the script will crash from trying to check the cell while it is still being edited by the user.

Excel as an improvised user interface

1. Script creates Excel worksheet for interface.
2. Script pauses while user enters parameters in worksheet.
3. When cell A2 is filled out, script resumes.
 - reads parameters from Excel and use as inputs
4. Script saves worksheet to document parameters used in analysis



7

This slide shows a scenario in which an Excel spreadsheet is used as an improvised interface for Python.

When the script is run, it creates the spreadsheet. The spreadsheet prompts the user for information and directs the user to enter a value of 1 in cell A2 when finished. The script periodically checks for a value of 1 in cell A2 and when it is received, the script saves the spreadsheet, to keep a record of the parameters, and proceeds with the analysis. The remainder of this video will discuss how to format a worksheet.

Font properties (read/edit)

- Applies to cell(s), column(s), or row(s)...
- Set font name...
`worksheet.Columns("A:Z").Font.Name = "Arial"`
- Set font to italic, bold, or underline...
`worksheet.Columns("A:Z").Font.Italic = True`
`worksheet.Columns("A:Z").Font.Bold = True`
`worksheet.Columns("A:Z").Font.Underline = True`
- Set font size...
`worksheet.Columns("A:Z").Font.Size = 12`

8

Formatting properties can be applied to cell, column, or row objects. The examples in the next few slides will use a column object.

The font name can be set through **font object** retrieved from the column, row, or cell object. Notice that multiple steps are combined into a single statement – the creation the column object, the retrieval of the font object, and the setting of the font object's name property.

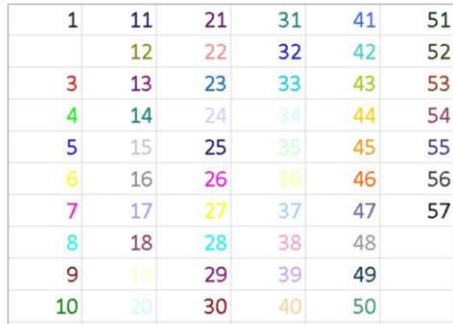
The font can be set to italics, bold, or underline by setting the appropriate font properties equal to True.

The font size can be specified.

Font color (read/edit)

- Set font color property...

```
worksheet.Columns("A:Z").Font.ColorIndex = 3
```



1	11	21	31	41	51
	12	22	32	42	52
3	13	23	33	43	53
4	14	24	34	44	54
5	15	25	35	45	55
6	16	26	36	46	56
7	17	27	37	47	57
8	18	28	38	48	
9	19	29	39	49	
10	20	30	40	50	

- Tip: index can also be obtained by using Python to read font color index property from an existing Excel file.

9

The font color can be specified by setting the **ColorIndex** to the code corresponding to the desired color.

This chart indicates the code number for each available font color.

Another way to determine the code number is to set the desired color for a cell in Excel and then read the ColorIndex property for the cell in Python.

Column width, row height, decimal places, text orientation

- Set column width...

```
worksheet.Columns("A").ColumnWidth = 20
```

- Set row height...

```
worksheet.Rows("1").RowHeight = 20
```

- Set number of decimal places... 54647.327

```
worksheet.Range("A1").NumberFormat = "0.000"
```

- Set text orientation...

```
worksheet.Range("A1").Orientation = -4171
```

```
worksheet.Range("A1").Orientation = -4170
```

1
class1
Class 1

10

The **ColumnWidth** can be set for a column object.

The **RowHeight** can be set for a row object.

The **NumberFormat** can be set to control the number of digits after the decimal. In the format string, each zero after the decimal corresponds to 1 digit.

The **Orientation** of the cell value(s) can be changed to vertical as shown in these examples.

Text formatting properties

- Set horizontal alignment...

`worksheet.Range("A1:A5").HorizontalAlignment = 3`

Right (4)
Left (2)
Center (3)

- Set vertical alignment ...

`worksheet.Range("A1:A5").VerticalAlignment = 2`

Top (1)
Center (2)
Bottom (3)

- Allow text to wrap...

`worksheet.Range("A1:A5").WrapText = True`

- Merge cells...

`worksheet.Range("B1:B5").MergeCells = True`

11

The **HorizontalAlignment** of cells, rows, or columns can be set to right, left, or center justification.

The **VerticalAlignment** can be set to top, center, or bottom alignment.

The **WrapText** property allows the text in a cell to wrap around to multiple lines.

The **MergeCells** property will combine the cells in the specified range into one cell.

Background color (read/edit)

- Can set for cell(s), row(s), and column(s)...
- Set background color property...

```
worksheet.Columns("A:Z").Interior.Color = 255
```



18711680	255	9868950	10092441	10066943
65535	10027212	39423	6750156	16776960
32768	0	16777215	16764057	153

- Tip: to get index value for a different color, use Python to read interior color property from an existing Excel file.

12

The background color of cells, rows, and columns can be set through the **Interior** object's **Color** property.

The code number that corresponds to the available colors are indicated in the table.

An alternate method to get the code number is to set the cell color in Excel and then use Python to read the Interior object's color property for the cell.

Example: formatting a worksheet

set column width...

```
wksheet.Columns("A:C").ColumnWidth = 20
```

set font properties...

```
wksheet.Range("A4:C4").Font.Size = 12  
wksheet.Range("A4:C4").Font.Italic = True  
wksheet.Range("A7:C7").Font.Bold = True  
wksheet.Range("A1").Font.ColorIndex = 3
```

set background colors...

```
wksheet.Range("A2").Interior.Color = 65535  
wksheet.Range("A8:C18").Interior.Color = 13434828
```

13

This slide will show an example script that formats a worksheet – the result of the scrip will be shown on the following slide. The first statement sets the column widths to 20 for columns A, B, and C.

The next set of statements sets the font properties for one or more cells:

- the size is set to 12 for cells A4 through C4,
- the style is set to italics for cells A4 through C4
- The style is set to bold for cells A7 through C7
- The font color code is set to 3, for cell A1, which corresponds to a red.

The final set of statements sets the colors of the cells.

- The color code of cell A2 is set to a yellow color (i.e. code 65535)
- The color code of cells A8 through C18 is set to a light green color (i.e. 13434828)

Example script result

	A	B	C	D	E
1	Type 1 in cell below and hit enter to continue...				
2					
3					
4	Slope con Max slope DEM file				

before code on previous slide

	A	B	C
1	Type 1 in cell below and hit enter to continue...		
2			
3			
4	<i>Slope constraint</i>	<i>Max slope (%)</i>	<i>DEM file</i>
5	slope		
6			
7	Other constraints	Buffer (in feet)	File name
8			
9			
10			

after code on previous slide

14

This slide shows the results from the example script on the previous slide. The unformatted spreadsheet was used as the input to the script.

The script formatted the spreadsheet as shown here.