Lisbon School of Economics & Management
Universidade de Lisboa

Excel & Python Analysis

2026/27

Introduction to Programming

Programming in Python

Objectives of this module

- Question: I'm not a software engineer! Why am I learning programming?
- Answers:
  1. Helps you train analytical thinking
  2. Software is the language of the world

     "*In the future, not knowing the language of computers will be as challenging as being illiterate or innumerate are today*"
  3. Promotes Computational Thinking

     "*teaches you how to tackle large problems by breaking them down into a sequence of smaller, more manageable problems*"

https://www.theguardian.com/technology/2014/feb/07/year-of-code-dan-crow-songkick

Objectives of this module (cont.)

- Q: Why Python?
- Answers:
  1. Easy to learn programming language
     - Interpreted
     - High Level
     - Multi-paradigm
  2. Widely available
     - No licensing fees
     - Several different freeware environments (IDE's)
     - Fastest growing programming language

How to Run Python Code

- There are several alternatives
- Online, command-line environments (shell)
  - Write the code and run it right away:
  - https://www.python.org/shell/
  - https://www.onlinegdb.com/online_python_interpreter
- Interactive Development Environments (IDE)

PyCharm
Spyder
Thonny
Visual Studio Code + Python Extension
Wing IDE
Eclipse + PyDev
Stani's Python Editor

How to Run Python Code

- Notebook Environments
  - Online - https://colab.research.google.com/
  - Jupyter Notebook (the most widely used)
    - Locally installed (runs on your browser with a local server software or inside an IDE)

Different programming paradigms

- Functional Programming
  - e.g.: Haskel and, to some extent, Excel functions
- Declarative Programming
  - e.g.: SQL
- Structured Procedural Programming
  - e.g: C, Pascal, **PYTHON**
- Object-Oriented Programming (OOP)
  - e.g.: C++, Java, C#, **PYTHON**

Python OO vs Structured Procedural

$$C_k^n = \frac{n!}{k!\,(n-k)!}$$

```python
class Combination:
    def __init__(self, n, k):
        self.n = n
        self.k = k

    def factorial(self, num):
        if num == 0:
            return 1
        else:
            return num * self.factorial(num - 1)

    def calculate(self):
        return self.factorial(self.n) / (self.factorial(self.k) * self.factorial(self.n - self.k))

# Create an instance of the Combination class
comb = Combination(10, 3)

# Call the calculate method
print(comb.calculate())
```

Preferred method for Software Engineers

```python
#Define the Factorial function
def fact(n):
    if n == 1:
        return n
    else:
        return n * fact(n-1)

#Define the Combinations function
def combination(n, k):
    return fact(n)/(fact(k) * fact(n-k))

# Call the calculate function
print(combination(10,3))
```

Preferred method for Data Scientists

Different forms of programming in Python

- “.py” script files:
  - Text files containing Python Code
  - Run from the command line: “`python file_name.py`”
  - Executed as a whole
  - **Ideal for creating modules or scripts**
- “.ipynb” Jupyter Notebook files:
  - Run in a Jupyter Notebook environment
  - JSON files with cells that can contain Code or Markdown
  - Facilitates iterative and interactive development
  - Allows execution of code blocks (cells)
  - **Ideal for exploratory data analysis and machine learning**

For this course

- Python
- Structured Procedural Programming
- Jupyter Notebook Environment

Lisbon School of Economics & Management
Universidade de Lisboa

Excel & Python Analysis

Variables
Operators

Programming in Python

What we are going to learn

Values and Variables:

- Integers
- Float
- Strings
- Boolean

Lists[], Sets{}

Conditional Structures:

- IF
- IF / ELSE
- IF / ELIF / ELSE

Cycles:

- FOR using:
  - list
  - set
  - range
- While:
  - using “Break”

Functions → “*def*”

Modules → import

Variables

- A Variable is a container that will hold a value
- Each container will have:
  - NAME → how you refer to it
  - TYPE → what type of data values it will contain
- Primitive types:
  - Integer (e.g. 123)
  - Floating Point (e.g. 123.456)
  - String (e.g. “This is a text” or ‘this is also a text’)
  - Boolean (**True** or **False**)

Creating a Variable

- In Python we create a variable with the assignment operator " **=** "
- The simple command:

  a = 10

  – Will do the following:
    - Create a container (variable)
    - Label the container with "**a**"
    - The container will have the type "Integer"
    - Put the integer number 10 into the container
  – Likewise, "**b = 12.345**" will create another container, label it "**b**", assign it the type "Floating Point" and put the value 12.345 into it

Types of Variables

- Python recognizes the value assigned to a variable and gives it the correct type

  - a = 10 → a will be type “Int” (Integer)
  - b = 1.123 → b will be type “Float” (Floating Point)
  - c = “This is a text” → c will be type “Str” (String)
  - d = True → d will be type “Bool” (Boolean)

Numeric Operators

Operator	Description	Example	Result
+	addition	5 + 8	13
-	subtraction	90 - 10	80
*	multiplication	4 * 7	28
/	floating point division	7 / 2	3.5
//	integer (truncating) division	7 // 2	3
%	modulus (remainder)	7 % 3	1
**	exponentiation	3 ** 4	81

(Lubanovic, 2014, p. 21)

Comparison Operators

- Equality ==
- inequality !=
- less than <
- less than or equal <=
- greater than >
- greater than or equal >=
- membership in

Other Operators

- Logical:
  - and
  - or
  - Not
- Assignment
  - = a = 5
  - += a += 5 ⇔ a = a + 5
  - -= a -=5 ⇔ a = a - 5
  - *= a *= 5 ⇔ a = a * 5
  - /= a /= 5 ⇔ a = a / 5
  - %= a %= 5 ⇔ a = a % 5
  - **= a **= 5 ⇔ a = a ** 5

String

- What is a string?
  - Sequence of characters:
    - “this is a string”
    - ‘this is also a string’
    - “can contain any char like 123 ! * etc.”
  - Printing a string:
    - print(“can contain any char like 123 ! * etc.”)
  - Printing several strings:
    - print("several", 'strings', 'in', "a", "sequence")

Special characters

- Escape Character: "\"
  - Means the next character has special meaning
- Some examples:
  - "\n" means "New Line"

```
print('A man,\nA plan,\nA canal:\nPanama.')
```

```
A man,
A plan,
A canal:
Panama.
```

  - "\t" means "Tab"

```
print("First \t1\nSecont\t2\nThird\t3")
```

```
First   1
Secont  2
Third   3
```

Special characters (cont)

- Substitution Character: "**%**" (older usage for printing)
  - When printing, put something where this char is
- Example:

```
print("today's date is %d of %s of the year %d" % (20, "January", 2020))
```

```
today's date is 20 of January of the year 2020
```

- Duplicating a char will remove its special meaning:

```
print('the discount will be %.2f%%' % 12.5 )
```

```
the discount will be 12.50%
```

Special characters (cont)

- Some more examples:

In:	myStr = "Today's date is %d of %s of the year %d" print(myStr % (20, "January", 2020))
Out:	Today's date is 20 of January of the year 2020

In:	myStr = "To print a \"Backslash\" we can duplicate the char '\\'" print(myStr)
Out:	To print a "Backslash" we can duplicate the char '\'

Formating strings (new version)

- Simple use of ".format()" or f'string'

```
v_day = 20
v_month = 'January'
v_year = 2022

print( f'The date is {v_day} of {v_month} of {v_year}')

print( 'The date is {} of {} of {}'.format(v_day, v_month, v_year) )
```

```
The date is 20 of January of 2022
The date is 20 of January of 2022
```

Formating strings (new version)

- More string formatting

```
print('This is a text ||{0:<20s}|| - ||{1:^20s}|| - ||{2:>20s}|| -- End'.format('left', 'middle align', 'Right'))
```

```
This is a text ||left                || - ||    middle align    || - ||               Right|| -- End
```

```
print('This is an integer ||{2:<10d}||{1:^10d}||{0:>10d}||'.format(10, 20, 30))
```

```
This is an integer ||30        ||    20    ||        10||
```

```
print("||{:15,.3f}||".format(1234.5))
print("||{:015,.3f}||".format(1234.5))
print("||{:^15,.3f}||".format(1234.5))
print("||{:^15,f}||".format(1234.5))
```

```
||      1,234.500||
||000,001,234.500||
||   1,234.500   ||
|| 1,234.500000  ||
```

Some String functions

In:	myStr = " This is the string to Play With " print("[“ + myStr.lower() + "]“)
Out:	[this is the string to play with]

In:	print("["+myStr.upper()+"]")
Out:	[THIS IS THE STRING TO PLAY WITH]

In:	print("["+myStr.strip()+"]")
Out:	[This is the string to Play With]

In:	print("I've removed %d spaces" % (len(myStr)-len(myStr.strip())))
Out:	I've removed 6 spaces

In:	print("["+myStr.replace("Play", "Work")+"]")
Out:	[This is the string to Work With]

In:	print("["+myStr.replace("Play", "Work").strip()+"]")
Out:	[This is the string to Work With]

More string functions

In:	myStr = "0123456789abcdefghi" print(myStr[9])
Out:	9

In:	print(myStr[10]) #the 10th char, counting from 0 (first position)
Out:	a

In:	print(myStr[3:11]) #from position 3 to position 11 excluding 11th
Out:	3456789a

In:	print(myStr[10:]) # from 10th position onward
Out:	abcdefghi

In:	print(myStr[:10]) # from the beginning til 10th position
Out:	0123456789

In:	print(myStr[-5:]) #the last 5 letters
Out:	efghi

Lisbon School of Economics & Management
Universidade de Lisboa

Excel & Python Analysis

Lists, Sets

Programming in Python

Lists, Sets

- How to create each structure?
  - myList = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
  - mySet = {'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday'}\

Lists

- What is a List (Object of Class “List”)
  - Ordered collection of items
  - Can have any number of elements and of different data types
    - A list can have another list as an element
  - It is possible to search, add, and remove items from the list
    - Mutable data type because it can be altered by adding or removing elements
- E.g.:
  - myList = ['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday‘]

Working with Lists – Some methods

```
In [1]: myList = [ 'Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday' ]
        print( myList )
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

```
In [2]: myList.append('Saturday') #Append a new value
        print(myList)
```

```
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday', 'Saturday']
```

```
In [3]: print(myList[3]) #Print a specific item (4th in this case - 0, 1, 2, 3)
```

```
Thursday
```

```
In [4]: lastItem = myList.pop()  #Remove the last item from the list
        print(lastItem)
        print(myList)
```

```
Saturday
['Monday', 'Tuesday', 'Wednesday', 'Thursday', 'Friday']
```

```
In [5]: print(myList[:3])  #Print the first 3 items
```

```
['Monday', 'Tuesday', 'Wednesday']
```

Operations (methods) with List objects

- append() – Adds an element to the end of the list
- clear() – Removes all elements from the list
- copy() – Returns a shallow copy of the list
- count() – Returns the total number of items passed as an argument
- extend() – Adds all elements of a list to some other list
- index() – Returns the index of an element (Note: If the same element appears multiple times in the list, then the index of the very first match is returned)
- insert() – Inserts an element to the list at the defined index
- pop() – Eliminates and returns an element from the list
- remove() – Eliminates an element from the list
- reverse() – Reverses the order of all elements of the list
- sort() – Sort all elements of a list in the ascending order

Sets

- What is a Set?
  - An unordered collection of simple objects in Python
  - It is mutable
  - Has no duplicate elements
  - Can have any number of elements and of different data types
  - Allow testing for membership, checking whether a set is a subset of some other set and finding the intersection between two sets
    - Mathematical Set Theory

Operations (methods) with Set Objects

- add() – Adds an item to the set
  - Note: As sets don't have repeating values, the item that is to be added to a set must not be already a member of the set.
- clear() – Removes all items of the set
- difference() – Returns a set with all elements of the invoking set but not of the second set
- intersection() – Returns an intersection of two sets
- union() – Returns a union of two sets

Working with sets

```
In [1]: oddNumbers = {1, 3, 5, 7, 9, 11, 13, 15, 17, 19}
        evenNumbers = {0, 2, 4, 6, 8, 10, 12, 14, 16, 18, 20}
        primeNumbers = {1, 2, 3, 5, 7, 11, 13, 17, 19}
```

```
In [2]: 3 in oddNumbers  #Check whether a number belongs to a set
```

```
Out[2]: True
```

```
In [3]: print(oddNumbers.difference(primeNumbers)) #Odd numbers that are not Prime
        print(evenNumbers.difference(primeNumbers)) #Even Numbers that are nor prime
```

```
{9, 15}
{0, 4, 6, 8, 10, 12, 14, 16, 18, 20}
```

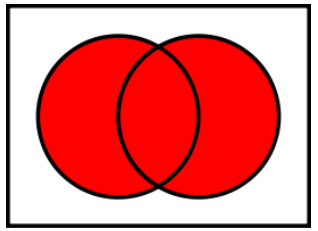
```
In [4]: print(oddNumbers.intersection(primeNumbers)) #Odd numbers that are also prime
        print(evenNumbers.intersection(primeNumbers))#Even Numbers that are also prime
```

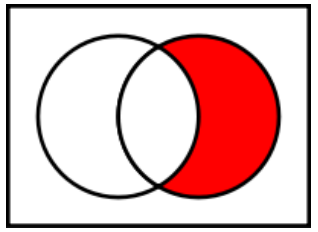
```
{1, 3, 5, 7, 11, 13, 17, 19}
{2}
```

```
In [5]: print(oddNumbers.union(primeNumbers)) #Numbers that are odd or prime
        print(evenNumbers.union(primeNumbers)) #numbers that are even or prime
```

```
{1, 2, 3, 5, 7, 9, 11, 13, 15, 17, 19}
{0, 1, 2, 3, 4, 5, 6, 7, 8, 10, 11, 12, 13, 14, 16, 17, 18, 19, 20}
```

More Set Examples

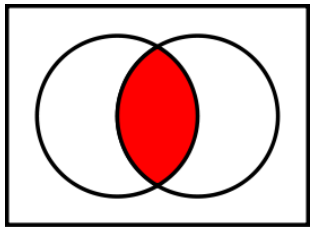


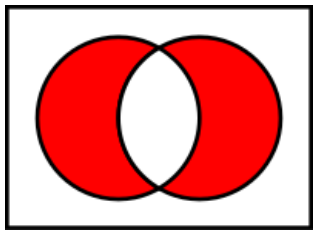


```
a = {1, 2, 3, 4}
b = {3, 4, 5, 6}

print (a.union(b))
print (a.intersection(b))
print (b.difference(a))
print (a.symmetric_difference(b))
```

```
{1, 2, 3, 4, 5, 6}
{3, 4}
{5, 6}
{1, 2, 5, 6}
```





Lisbon School of Economics & Management
Universidade de Lisboa

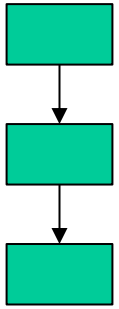
Excel & Python Analysis

Conditions
Cycles

Programming Control Structures

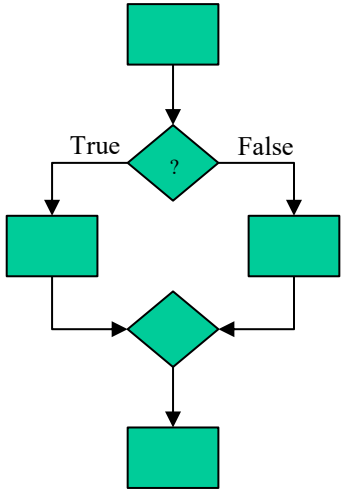
Programming Control Structures

Sequence



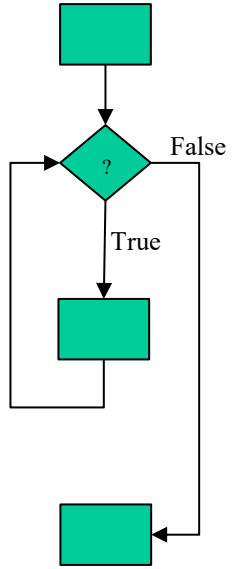
Normal flow of execution.
Each instruction is executed before the next

Decision



if
if / else
if / elif / else

Cycle



for: using lists, sets, tuples
for: using ranges
while

Decision

In [1]:
```
whatToTest = input("What do you want to test?:")

if whatToTest in "aeiouAEIOU":
    print("%s is a vowel" % whatToTest)
else:
    print("%s is NOT vowel" % whatToTest)
```

```
What do you want to test?:E
E is a vowel
```

In [2]:
```
whatToTest = input("What do you want to test?:")

if whatToTest in "aeiouAEIOU":
    print("%s is a vowel" % whatToTest)
elif whatToTest in "0123456789":
    print("%s is a number" % whatToTest)
else:
    print("%s is neither a vowel nor a number" % whatToTest)
```

```
What do you want to test?:X
X is neither a vowel nor a number
```

Cycle FOR

- Cycle “for”
  - List
  - set

In [1]:
```
myList = ["Mon", "Tue", "Wed", "Thu", "Fri"]
mySet = {"Mon", "Tue", "Wed", "Thu", "Fri"}
```

In [2]:
```
for i in myList:
    print(i)
```

```
Mon
Tue
Wed
Thu
Fri
```

In [3]:
```
for i in mySet:
    print(i)
```

```
Wed
Thu
Tue
Fri
Mon
```

Cycle FOR with “range”

- Use range(initial, final, *step*)
  - “step” is optional. If omitted, assumed “1”

Attention: The Range will be: Initial <= I < final

```
In [1]: for i in range(10, 20, 2):
            print(i)
```

```
10
12
14
16
18
```

```
In [2]: for i in range(30, 20, -2):
            print(i)
```

```
30
28
26
24
22
```

Cycle “While”

```
In [5]: count = 1
        while count <= 5:
            print(count)
            count += 1
```

```
1
2
3
4
5
```

```
In [1]: print("----------hellow----------")
        while True:
            stuff = input("String to capitalize [type q to quit]: ")
            if stuff == "q":
                print("----------goodbye----------")
                break
            print(stuff.capitalize())
```

```
----------hellow----------
String to capitalize [type q to quit]: Does it work?
Does it work?
String to capitalize [type q to quit]: yes it does
Yes it does
String to capitalize [type q to quit]: q
----------goodbye----------
```

Lisbon School of Economics & Management
Universidade de Lisboa

Excel & Python Analysis

Defining Functions
Importing Modules
Handling Exceptions

Programming in Python

Functions

- What is a function?
  - named piece of code, separate from all others
  - can take any number and type of input parameters
  - return any number and type of output results
- How to use a function?
  - Define it
  - Call it
- Syntax

  “ def «function_name» (list of arguments): ”

Example Function

A function to calculate the value of the factorial of a number:

```
In [1]:
def fact(n):
    res = 1
    for i in range(2,n+1):
        res = res * i
    return res
```

A function may call itself (Recursive function):

```
In [1]:
def fact(n):
    if n == 1:
        return n
    else:
        return n * fact(n-1)

fact(10)
```

```
Out[1]: 3628800
```

Modules

- *module* is a file of Python code
- Import a module

  import «module-name» as «myName»

  optional
- Example:

```
In [1]: import random as r

In [2]: myList = [1, 5, 7, 23, 65, 32, 65, 49]

In [3]: print(r.choice(myList))
        23

In [4]: print(r.randint(100, 1000))
        723
```

Exception Handler

- Protect your program with “**try**” / “**except**”:

In [2]:

```
while True:
    try:
        num = eval(input('Give me a number: '))
        if num%2 == 0:
            print('even number')
        else:
            print('odd number')
    except:
        x = input('Wrong input! Continue? ("Y/N")')
        if x == 'N' or x == 'n':
            break
```

List Comprehensions

- Concise way of creating lists in Python.
- Creates a list by generating new elements in a range, filtering out elements based on a certain condition.
- The basic syntax for a list comprehension is as follows:

```
[expression for item in iterable if condition]
```

  - `expression` is the operation to be performed on each element in the iterable.
  - `item` is a variable that represents each element in the iterable.
  - `iterable` is any Python object that can be iterated over

List Comprehensions exemples

```
squared_numbers = [x**2 for x in range(10)]
print(squared_numbers)
```

✓ 0.4s

```
[0, 1, 4, 9, 16, 25, 36, 49, 64, 81]
```

```
even_numbers = [x for x in [1, 2, 3, 4, 5, 6, 7, 8, 9, 10] if x % 2 == 0]
print(even_numbers)
```

✓ 0.3s

```
[2, 4, 6, 8, 10]
```

```
def is_even(x):
    return x % 2 == 0

even_numbers = [x for x in range(1, 11) if is_even(x)]
print(even_numbers)
```

✓ 0.5s

```
[2, 4, 6, 8, 10]
```

Lisbon School of Economics & Management
Universidade de Lisboa

Excel & Python Analysis

Handling Tables and Files

Programming in Python

Pandas

- Pandas
  - Very powerful, flexible and easy to use open-source data analysis and manipulation module
  - Some useful functionality:
    - Read and write Comma Separated Values (“.csv”) files
    - Read and Write Excel files
    - Handle tables (data frames)

```
In [ ]: import pandas [as pd]
```

optional

Data Frame

- 2-dimensional data structure
- columns of potentially different types
- Similar to an Excel table
- The most commonly used pandas object

Read ".csv" file

```
stdNo   stdName BirthDate   email   Test 1  Test 2  Lab 1   Lab 2   Lab 3   Lab 4
701 Rosa C. Olsen   07/06/1996  l0701@aln.iseg.ulisboa.pt   16,8    9,2 12,6    10,6    13,7    9,2
798 Clarence Sheridan   03/01/1996  l0798@aln.iseg.ulisboa.pt   15,8    15,8    18,9    19,6    14,2    4,3
854 Jaclyn T. Glover    20/09/1995  l0854@aln.iseg.ulisboa.pt   11,2    8,5 9,1 16,8    13  19,6
663 Catherine U. Buggs  09/12/1995  l0663@aln.iseg.ulisboa.pt   15  6,8 5,4 8,2 6,2 4,4
921 James E. Watkins    08/08/1999  l0921@aln.iseg.ulisboa.pt   11  8   16,9    5,4 14,9    5,1
890 Babara T. Best  09/04/1998  l0890@aln.iseg.ulisboa.pt   14  12  19,3    11,5    18,9    19,1
666 Carlton L. Dickerson    24/07/1997  l0666@aln.iseg.ulisboa.pt   15,7    7,1 19,7    8,5 8,7 4,3
843 Kerry K. Ratliff    30/05/2000  l0843@aln.iseg.ulisboa.pt   11,4    8,6 14,6    12,1    13,4    19,8
991 Carrie D. Johnson   21/08/1998  l0991@aln.iseg.ulisboa.pt   8,2 11,3    19,1    14,9    13,8    12
798 Harold Brown    18
713 Camilla N. Welch
755 Chelsey K. Kinley
786 Julie Triplett  15
507 John Lopez  29/12/
873 James O. Boyce  24
581 Pedro J. Taylor 17
```

grades.csv file
Separator: "Tab"

```
In [1]: import pandas as pd

In [2]: my_dataframe = pd.read_csv("grades.csv", sep='\t', encoding = 'utf8')

In [3]: my_dataframe.head()
```

Out[3]:

	stdNo	stdName	BirthDate	email	Test 1	Test 2	Lab 1	Lab 2	Lab 3	Lab 4
0	701	Rosa C. Olsen	07/06/1996	l0701@aln.iseg.ulisboa.pt	16,8	9,2	12,6	10,6	13,7	9,2
1	798	Clarence Sheridan	03/01/1996	l0798@aln.iseg.ulisboa.pt	15,8	15,8	18,9	19,6	14,2	4,3
2	854	Jaclyn T. Glover	20/09/1995	l0854@aln.iseg.ulisboa.pt	11,2	8,5	9,1	16,8	13	19,6
3	663	Catherine U. Buggs	09/12/1995	l0663@aln.iseg.ulisboa.pt	15	6,8	5,4	8,2	6,2	4,4
4	921	James E. Watkins	08/08/1999	l0921@aln.iseg.ulisboa.pt	11	8	16,9	5,4	14,9	5,1

Accessing data-frame cells with ".loc"

- There are different methods to access a data-frame cell
  - "loc", with the row number and column name(s):

```
my_dataframe.loc[line_num, col_name]
```

```
my_dataframe.loc[line_i:line_f, col_i:col_f]
```

  - The .loc method is inclusive for line and column ranges

Accessing data-frame cells with ".iloc"

– "iloc", with the row number and column number:

```
my_dataframe.iloc[line_num, col_num]
```

```
my_dataframe.iloc[line_i:line_f, col_i:col_f]
```

– The .loc method follows standard Python rules for line and column ranges

Writing a data-frame to Excel

- Once having a data-frame, we can write it to an Excel file
  - Create a container to write the Excel file:

In [ ]:
```
my_writer = pd.ExcelWriter('myNewExcelFile.xlsx', engine='xlsxwriter')
```

  - Write the data-frame to the container “my_writer”, with a worksheet called “Grades” (do not write the index)

In [ ]:
```
my_dataframe.to_excel(my_writer, sheet_name = 'Grades', index=False)
```

  - Save the Excel File

In [ ]:
```
my_writer.save()
```