

```

1823
1824     class DefinitionSpaTab(scrolledpanel.ScrolledPanel):
1825         """
1826         The tab of the interface, containing all the controls to enter the coefficient properties
1827         """
1828
1829     def __init__(self, parent):
1830         """
1831         Initializes the panel class and includes all controls, binds all functions, etc.
1832         :param parent: the parent frame
1833         :return:
1834         """
1835         super().__init__(parent=parent, id=1)
1836
1837         # Variables --> to be completed
1838
1839 LICAR1104: self.parent = parent
1840
1841         self.definition = self.DefinitionTab(DEFINITION)
1842
1843         # Buttons
1844         self.button = self.Button(OK)
1845         self.button = self.Button(CANCEL)
1846         self.button = self.Button(HELP)
1847         self.button = self.Button(ABOUT)
1848         self.button = self.Button(HELP)
1849         self.button = self.Button(ABOUT)
1850         self.button = self.Button(HELP)
1851         self.button = self.Button(ABOUT)
1852         self.button = self.Button(HELP)
1853         self.button = self.Button(ABOUT)
1854         self.button = self.Button(HELP)
1855         self.button = self.Button(ABOUT)
1856         self.button = self.Button(HELP)
1857         self.button = self.Button(ABOUT)
1858         self.button = self.Button(HELP)
1859         self.button = self.Button(ABOUT)
1860         self.button = self.Button(HELP)
1861         self.button = self.Button(ABOUT)
1862         self.button = self.Button(HELP)
1863         self.button = self.Button(ABOUT)
1864         self.button = self.Button(HELP)
1865         self.button = self.Button(ABOUT)
1866         self.button = self.Button(HELP)
1867         self.button = self.Button(ABOUT)
1868         self.button = self.Button(HELP)
1869         self.button = self.Button(ABOUT)
1870         self.button = self.Button(HELP)
1871         self.button = self.Button(ABOUT)
1872         self.button = self.Button(HELP)
1873         self.button = self.Button(ABOUT)
1874         self.button = self.Button(HELP)
1875         self.button = self.Button(ABOUT)
1876         self.button = self.Button(HELP)
1877         self.button = self.Button(ABOUT)
1878         self.button = self.Button(HELP)
1879         self.button = self.Button(ABOUT)
1880         self.button = self.Button(HELP)
1881         self.button = self.Button(ABOUT)
1882         self.button = self.Button(HELP)
1883         self.button = self.Button(ABOUT)
1884         self.button = self.Button(HELP)
1885         self.button = self.Button(ABOUT)
1886         self.button = self.Button(HELP)
1887         self.button = self.Button(ABOUT)
1888         self.button = self.Button(HELP)
1889         self.button = self.Button(ABOUT)
1890         self.button = self.Button(HELP)
1891         self.button = self.Button(ABOUT)
1892         self.button = self.Button(HELP)
1893         self.button = self.Button(ABOUT)
1894         self.button = self.Button(HELP)
1895         self.button = self.Button(ABOUT)
1896         self.button = self.Button(HELP)
1897         self.button = self.Button(ABOUT)
1898         self.button = self.Button(HELP)
1899         self.button = self.Button(ABOUT)
1900         self.button = self.Button(HELP)
1901         self.button = self.Button(ABOUT)
1902         self.button = self.Button(HELP)
1903         self.button = self.Button(ABOUT)
1904         self.button = self.Button(HELP)
1905         self.button = self.Button(ABOUT)
1906         self.button = self.Button(HELP)
1907         self.button = self.Button(ABOUT)
1908         self.button = self.Button(HELP)
1909         self.button = self.Button(ABOUT)
1910         self.button = self.Button(HELP)
1911         self.button = self.Button(ABOUT)
1912         self.button = self.Button(HELP)
1913         self.button = self.Button(ABOUT)
1914         self.button = self.Button(HELP)
1915         self.button = self.Button(ABOUT)
1916         self.button = self.Button(HELP)
1917         self.button = self.Button(ABOUT)
1918         self.button = self.Button(HELP)
1919         self.button = self.Button(ABOUT)
1920         self.button = self.Button(HELP)
1921         self.button = self.Button(ABOUT)
1922         self.button = self.Button(HELP)
1923         self.button = self.Button(ABOUT)
1924         self.button = self.Button(HELP)
1925         self.button = self.Button(ABOUT)
1926         self.button = self.Button(HELP)
1927         self.button = self.Button(ABOUT)
1928         self.button = self.Button(HELP)
1929         self.button = self.Button(ABOUT)
1930         self.button = self.Button(HELP)
1931         self.button = self.Button(ABOUT)
1932         self.button = self.Button(HELP)
1933         self.button = self.Button(ABOUT)
1934         self.button = self.Button(HELP)
1935         self.button = self.Button(ABOUT)
1936         self.button = self.Button(HELP)
1937         self.button = self.Button(ABOUT)
1938         self.button = self.Button(HELP)
1939         self.button = self.Button(ABOUT)
1940         self.button = self.Button(HELP)
1941         self.button = self.Button(ABOUT)
1942         self.button = self.Button(HELP)
1943         self.button = self.Button(ABOUT)
1944         self.button = self.Button(HELP)
1945         self.button = self.Button(ABOUT)
1946         self.button = self.Button(HELP)
1947         self.button = self.Button(ABOUT)
1948         self.button = self.Button(HELP)
1949         self.button = self.Button(ABOUT)
1950         self.button = self.Button(HELP)
1951         self.button = self.Button(ABOUT)
1952         self.button = self.Button(HELP)
1953         self.button = self.Button(ABOUT)
1954         self.button = self.Button(HELP)
1955         self.button = self.Button(ABOUT)
1956         self.button = self.Button(HELP)
1957         self.button = self.Button(ABOUT)
1958         self.button = self.Button(HELP)
1959         self.button = self.Button(ABOUT)
1960         self.button = self.Button(HELP)
1961         self.button = self.Button(ABOUT)
1962         self.button = self.Button(HELP)
1963         self.button = self.Button(ABOUT)
1964         self.button = self.Button(HELP)
1965         self.button = self.Button(ABOUT)
1966         self.button = self.Button(HELP)
1967         self.button = self.Button(ABOUT)
1968         self.button = self.Button(HELP)
1969         self.button = self.Button(ABOUT)
1970         self.button = self.Button(HELP)
1971         self.button = self.Button(ABOUT)
1972         self.button = self.Button(HELP)
1973         self.button = self.Button(ABOUT)
1974         self.button = self.Button(HELP)
1975         self.button = self.Button(ABOUT)
1976         self.button = self.Button(HELP)
1977         self.button = self.Button(ABOUT)
1978         self.button = self.Button(HELP)
1979         self.button = self.Button(ABOUT)
1980         self.button = self.Button(HELP)
1981         self.button = self.Button(ABOUT)
1982         self.button = self.Button(HELP)
1983         self.button = self.Button(ABOUT)
1984         self.button = self.Button(HELP)
1985         self.button = self.Button(ABOUT)
1986         self.button = self.Button(HELP)
1987         self.button = self.Button(ABOUT)
1988         self.button = self.Button(HELP)
1989         self.button = self.Button(ABOUT)
1990         self.button = self.Button(HELP)
1991         self.button = self.Button(ABOUT)
1992         self.button = self.Button(HELP)
1993         self.button = self.Button(ABOUT)
1994         self.button = self.Button(HELP)
1995         self.button = self.Button(ABOUT)
1996         self.button = self.Button(HELP)
1997         self.button = self.Button(ABOUT)
1998         self.button = self.Button(HELP)
1999         self.button = self.Button(ABOUT)
2000         self.button = self.Button(HELP)


```

LICAR1104: *Informatique et Méthodes Numériques*

Functions | Pandas

Prof. André Stephan  
Assistant Professor in Environmental Performance and Parametric Design






## Seminar plan

- Seminar learning outcomes
- Active lecture: functions
- Activity one: calling functions
- Active lecture: introduction to pandas
- Activity two: calculate the embodied energy of a building with pandas

## Seminar learning objectives

At the end of this seminar you will be able to:

- Define python and lambda functions
- Demonstrate a basic understanding of pandas dataframes and use them practically

def



## Functions (in python)

- Group of related statements that perform a given task
- Enables a program to be divided into smaller parts
- Action/Verb → Function (e.g. calculate)
- Avoids repetition!!! (remember the PEP08 recommendations)

## Functions in python – declaration - call

```
def function_name(argument_1, argument_2, ..., argument_n):
    """docstring"""
    statement_1
    statement_2
    ...
    statement_n
    (return var_1, var_2, ..., var_n)

def power_2(number):
    try:
        return number**2
    except TypeError:
        print("Couldn't produce power 2 of " + str(number) + "
              which is a " + str(type(number)))

power_2(3)
>>> 9
power_2([5])
TypeError "Couldn't produce power 2 of [5] which is a <class
'list'>"
```


 André Stephan | [andre.stephan@uclouvain.be](mailto:andre.stephan@uclouvain.be) [in/andrestephan](https://www.linkedin.com/in/andrestephan)

## Functions in python - docstring

```
def power_2(number):
    """This function tries to produce the power 2 of a given
    number and returns an error if it fails"""
    try:
        return number**2
    except TypeError:
        print("Couldn't produce power 2 of " + str(number) + "
              which is a " + str(type(number)))

print(power_2.__doc__)

>>> "This function tries to produce the power 2 of a given
number and returns an error if it fails"
```

 André Stephan | [andre.stephan@uclouvain.be](mailto:andre.stephan@uclouvain.be) [in/andrestephan](https://www.linkedin.com/in/andrestephan)

## Functions in python - return

```
def absolute_value(number):
    """This function tries to produce the power 2 of a given
    number and returns an error if it fails"""
    try:
        if number >=0:
            return number
        else:
            return -number
    except TypeError:
        print("Couldn't produce the absolute value of " +
              str(number) + " which is a " + str(type(number)))

absolute_value(-15)
>>>15
absolute_value(124)
>>>124
```

## Functions in python – good to know

```
def hi():
    print("Hi")
    return None

hi()
>>> Hi

hi
>>> <function hi at 0x00000246B47314C0 >
```

## Functions in python – scope

```
def test():
    a = 20
    print('Value of a inside function ', a)

a = 40
test()
>>>'Value of a inside function 20'

print(a)
>>> 40
```

## Lambda functions in python

- Short anonymous function
- Practical for very short expressions
- Multiply arguments, one expression

```
lambda arg_1,...,arg_n: <result to return>

z = lambda x,y: (x+y)**2
z(2,3)
>>>25
```





- *“pandas is a fast, powerful, flexible and easy to use open source data analysis and manipulation tool, built on top of the Python programming language” (pandas.org)*
- Started in 2008 (2009 open-access version)
- Wraps around numpy nd-**arrays**, turning them into **DataFrames** (numpy 2-d array on steroids) with a plethora of additional methods
- Dataframes can receive an index and column labels,



- Pandas DataFrame is its core object
- Stores different types of variables! (like python data structures)
- Basically it's a matrix with 2-dimensions, and 1-n indexes, most often 1
- A DataFrame can have names for its rows (indexes) and columns (labels)
- Axis 0 is ? **Rows**
- Axis 1 is ? **Columns**



```
import pandas as pd
df = pd.DataFrame({'one': [1,2,3,4], 'two': [5,6,7,8]})
>>> df
```

	one	two
0	1	5
1	2	6
2	3	7
3	4	8



	df	one	two
	0	1	5
	1	2	6
	2	3	7
	3	4	8


```
df['one']
>>>
   one
0    1
1    2
2    3
3    4

df.loc[0]
>>>
   one  two
0    1    5
Name: 0, dtype: int64

df.iloc[0]
>>>
   one  two
0    1    5
Name: 0, dtype: int64

df[['one', 'two']]
>>>
   one  two
0    1    5
1    2    6
2    3    7
3    4    8
```





```

df
  one  two
0    1    5
1    2    6
2    3    7
3    4    8

```

```

>>>df[0]
8

```

```

>>>df['one'][3]
array([49, 64, 81])

```

```

>>>df['two'][-1]
KeyError

```

```



>>>df - 2
  one  two
0    1    5
1    2    6


```



```

>>>df.loc[df['one']<=2]
  one  two
0   -1    3
1    0    4
2    1    5
3    2    6

```

UCLouvain  André Stephan | [andre.stephan@uclouvain.be](mailto:andre.stephan@uclouvain.be)  /andrestephan



```



df = pd.read_excel('filename.xlsx', index_col=0, header=0)

df = pd.read_csv('filename.csv', index_col=0, header=0)

```

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_excel.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_excel.html)

[https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read\\_csv.html](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.read_csv.html)

UCLouvain  André Stephan | [andre.stephan@uclouvain.be](mailto:andre.stephan@uclouvain.be)  /andrestephan



- Multiplying a dataframe by another:

```
df1.multiply(df2, axis='index')
```

<https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.multiply.html>

## Activity two: using pandas to calculate the embodied environmental flows of a building

- Download a reworked version of the original EPiC database <https://doi.org/10.26188/5dc228ef98c5a> as excel (.xlsx) from: <https://www.dropbox.com/s/0p5wfvjrmny8u8l/EPiC%20Database%202019.xlsx?dl=0>
- Put the file in your working python directory
- Read the excel file as a dataframe (named *epic\_df*), use only columns 2,3,4,5,6, use column 0 as the index\_col, use excel's row 6 as the header
- Download the bill of quantities of Belgian Passive House from: [https://www.dropbox.com/s/cxyp2wd8hgozdns/ph\\_boq.xlsx?dl=0](https://www.dropbox.com/s/cxyp2wd8hgozdns/ph_boq.xlsx?dl=0)
- Read the excel file as a dataframe (named *boq*), use column 0 as the index\_col, use row 0 as the header
- Calculate the embodied energy, water and greenhouse gas emissions of each material in the building (**note: materials in the boq are all in the correct units**)
- Calculate the embodied energy, water and greenhouse gas emissions of the whole building.

15 mins

15 mins

## Activity two: using pandas to calculate the embodied environmental flows of a building

- Read the excel file as a dataframe (named `epic_df`), use columns 2,3,4,5,6, use column 0 as the `index_col`, use excel's row 6 as the header

```
import pandas as pd
epic_df = pd.read_excel('EPiC Database 2019.xlsx',
header=5, usecols=[2,3,4,5,6], index_col=0)
```

## Activity two: using pandas to calculate the embodied environmental flows of a building

- Read the excel file as a dataframe (named `boq`), use column 0 as the `index_col`, use row 0 as the header

```
boq = pd.read_excel('ph_boq.xlsx', index_col=0)
```

## Activity two: using pandas to calculate the embodied environmental flows of a building

- Calculate the embodied energy, water and greenhouse gas emissions of each material in the building

Keeps only the relevant columns for the calculation




Filters out all unnecessary materials from the database

```
res =
epic df.loc[boq.index][['EE', 'EW', 'EGHG']]
.multiply(boq['Delivered quantity'],
axis=0)
```

Multiplies by the quantities of materials in the building

## Activity two: using pandas to calculate the embodied environmental flows of a building

- Calculate the embodied energy, water and greenhouse gas emissions of the whole building

`sum(res.EE)`  1971319.8 MJ = 1 971 GJ  
`sum(res.EW)`  4747755.4 L = 47 478 kL  
`sum(res.EGHG)`  150339.3 kgCO<sub>2</sub>e = 150 tCO<sub>2</sub>e

```
sum(res['EE'])
sum(res['EW'])
sum(res['EGHG'])
```

## What if you had to write a function that...

- Receives a boq file formatted like the one you just saw
- Automatically calculates the Embodied energy, water and greenhouse gas emissions
- Returns a dataframe with materials as the index and three columns of EE, EW, and EGHG as well as a dictionary containing IEE, IEW, and IEGHG as keys, and the total initial embodied energy, water and greenhouse gas emissions as values

## Function that calculates initial embodied flows from the EPiC database for a given boq file

```
def calc_embodied_flows(boq_file:str,epic_db_file_path:str =
'EpiC Database 2019.xlsx'):
    epic_df = pd.read_excel(epic_db_file_path, header=5,
        usecols=[2,3,4,5,6], index_col=0)
    boq = pd.read_excel(boq_file, index_col=0)
    res =
    epic_df.loc[boq.index][['EE','EW','EGHG']].multiply(boq['D
    elivered quantity'], axis=0)
    total_ef = {flow: sum(res[flow]) for flow in
    ['EE','EW','EGHG']}
    return res, total_ef
```

```
a,b = calc_embodied_flows('ph_boq.xlsx')
```

Next time: Week 12

Basics of:

# Object-oriented programming