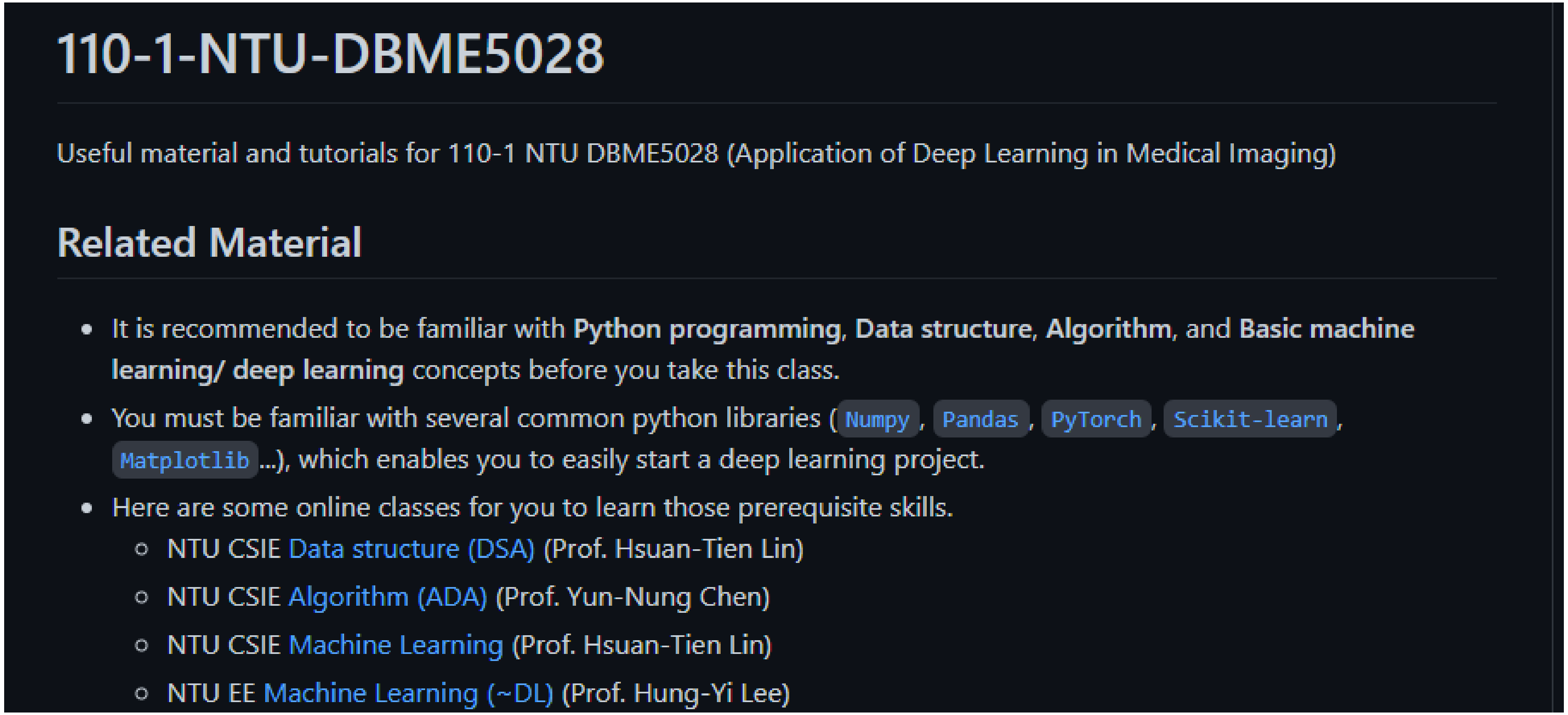
110-1 NTU DBME5028

Application of Deep Learning in Medical Imaging

Pre-test explanation

2021/10/06

TA M.Y. Ho



110-1-NTU-DBME5028

Useful material and tutorials for 110-1 NTU DBME5028 (Application of Deep Learning in Medical Imaging)

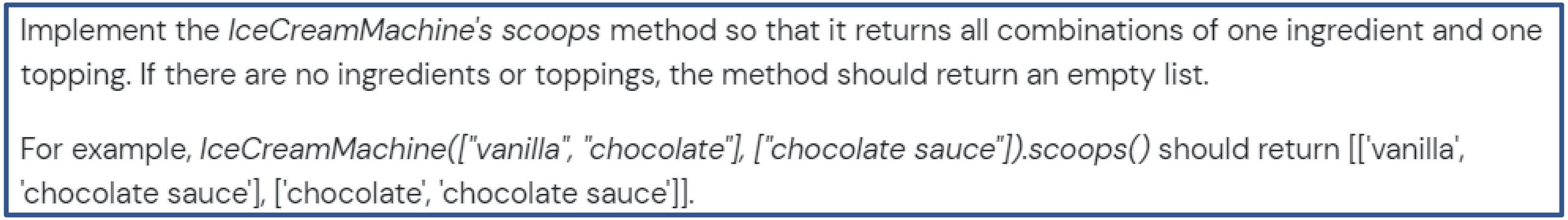
Related Material

- It is recommended to be familiar with **Python programming**, **Data structure**, **Algorithm**, and **Basic machine learning/ deep learning** concepts before you take this class.
- You must be familiar with several common python libraries (`Numpy` , `Pandas` , `PyTorch` , `Scikit-learn` , `Matplotlib` ...), which enables you to easily start a deep learning project.
- Here are some online classes for you to learn those prerequisite skills.
  - NTU CSIE Data structure (DSA) (Prof. Hsuan-Tien Lin)
  - NTU CSIE Algorithm (ADA) (Prof. Yun-Nung Chen)
  - NTU CSIE Machine Learning (Prof. Hsuan-Tien Lin)
  - NTU EE Machine Learning (~DL) (Prof. Hung-Yi Lee)

https://github.com/Kaminyou/110-1-NTU-DBME5028

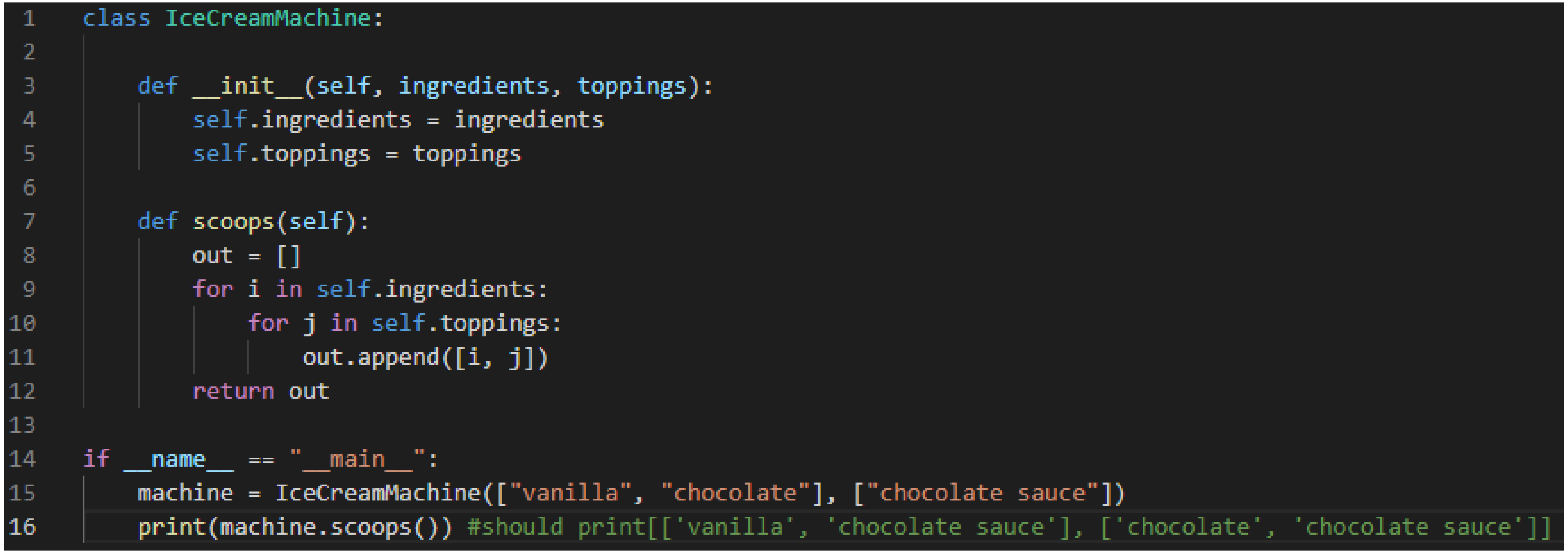
#1

Ice Cream Machine



Implement the *IceCreamMachine's scoops* method so that it returns all combinations of one ingredient and one topping. If there are no ingredients or toppings, the method should return an empty list.

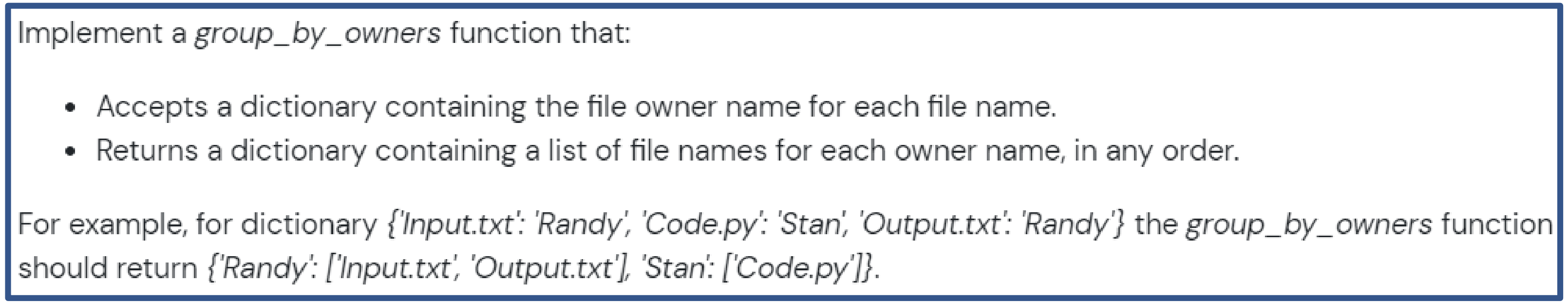
For example, *IceCreamMachine(["vanilla", "chocolate"], ["chocolate sauce"]).scoops()* should return [['vanilla', 'chocolate sauce'], ['chocolate', 'chocolate sauce']].



```python
class IceCreamMachine:

    def __init__(self, ingredients, toppings):
        self.ingredients = ingredients
        self.toppings = toppings

    def scoops(self):
        out = []
        for i in self.ingredients:
            for j in self.toppings:
                out.append([i, j])
        return out

if __name__ == "__main__":
    machine = IceCreamMachine(["vanilla", "chocolate"], ["chocolate sauce"])
    print(machine.scoops()) #should print[['vanilla', 'chocolate sauce'], ['chocolate', 'chocolate sauce']]
```

#2

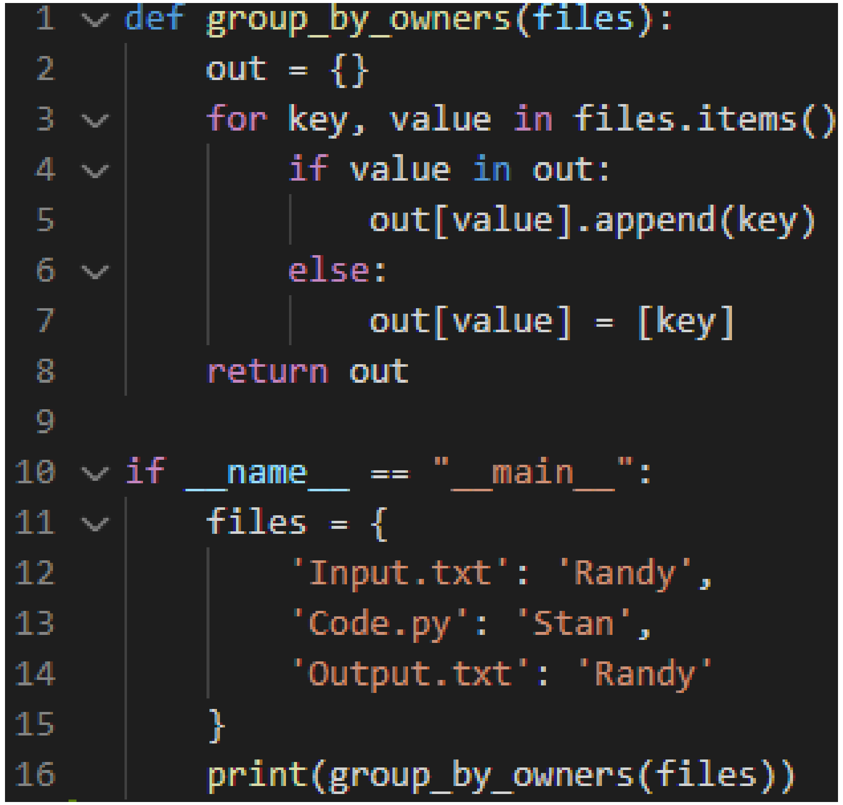
File Owners



Implement a *group_by_owners* function that:

- Accepts a dictionary containing the file owner name for each file name.
- Returns a dictionary containing a list of file names for each owner name, in any order.

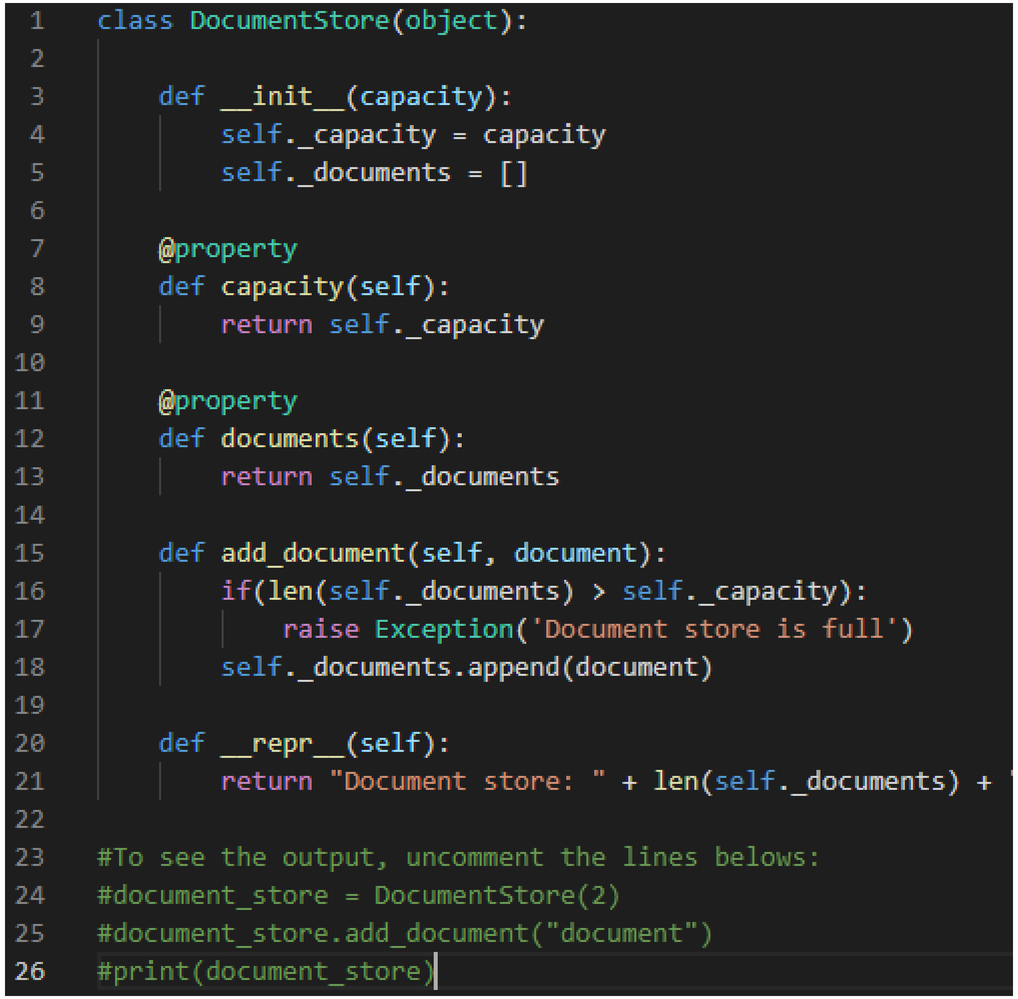
For example, for dictionary *{'Input.txt': 'Randy', 'Code.py': 'Stan', 'Output.txt': 'Randy'}* the *group_by_owners* function should return *{'Randy': ['Input.txt', 'Output.txt'], 'Stan': ['Code.py']}*.



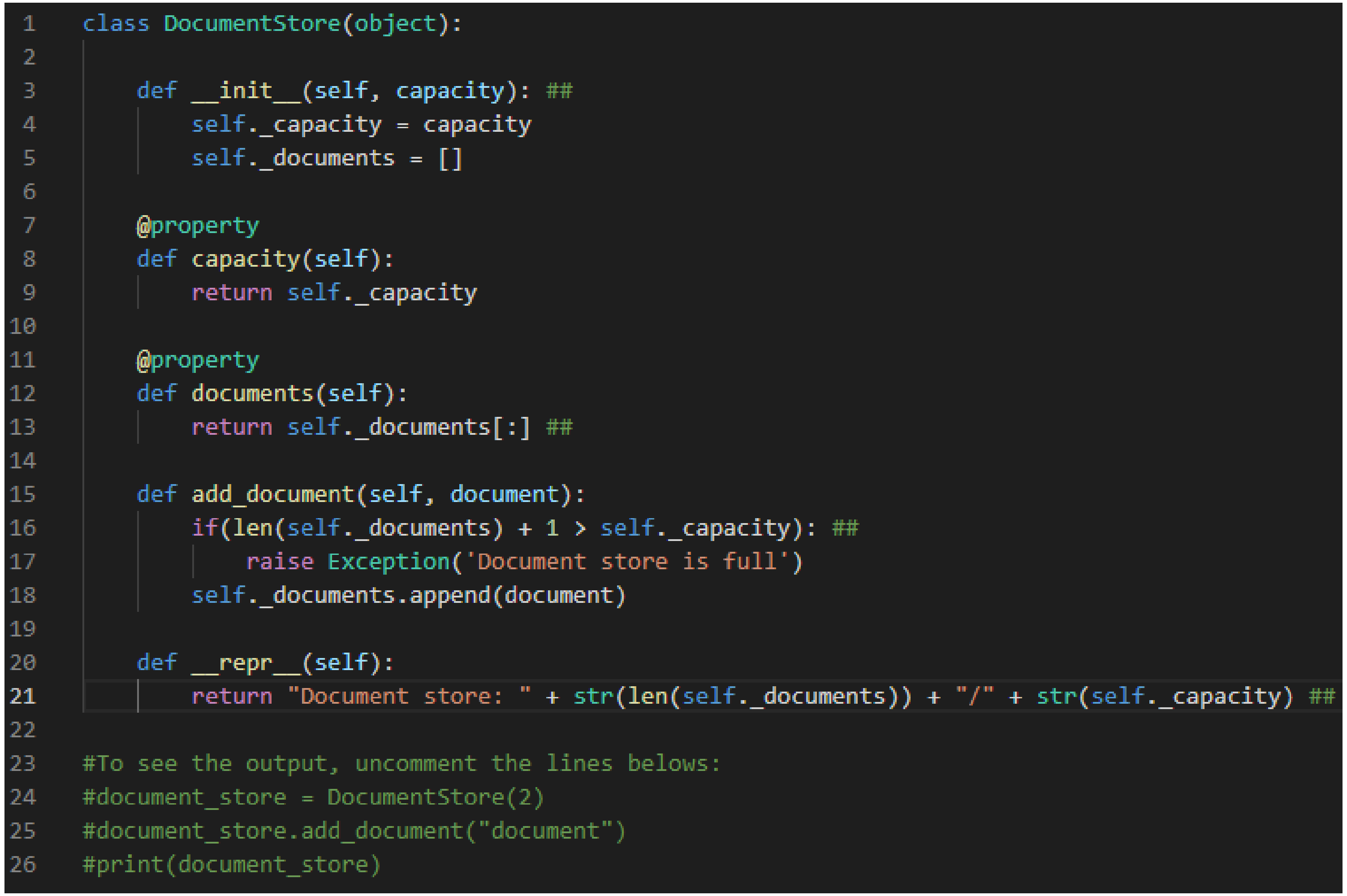
```
def group_by_owners(files):
    out = {}
    for key, value in files.items()
        if value in out:
            out[value].append(key)
        else:
            out[value] = [key]
    return out

if __name__ == "__main__":
    files = {
        'Input.txt': 'Randy',
        'Code.py': 'Stan',
        'Output.txt': 'Randy'
    }
    print(group_by_owners(files))
```

#3

Document Store



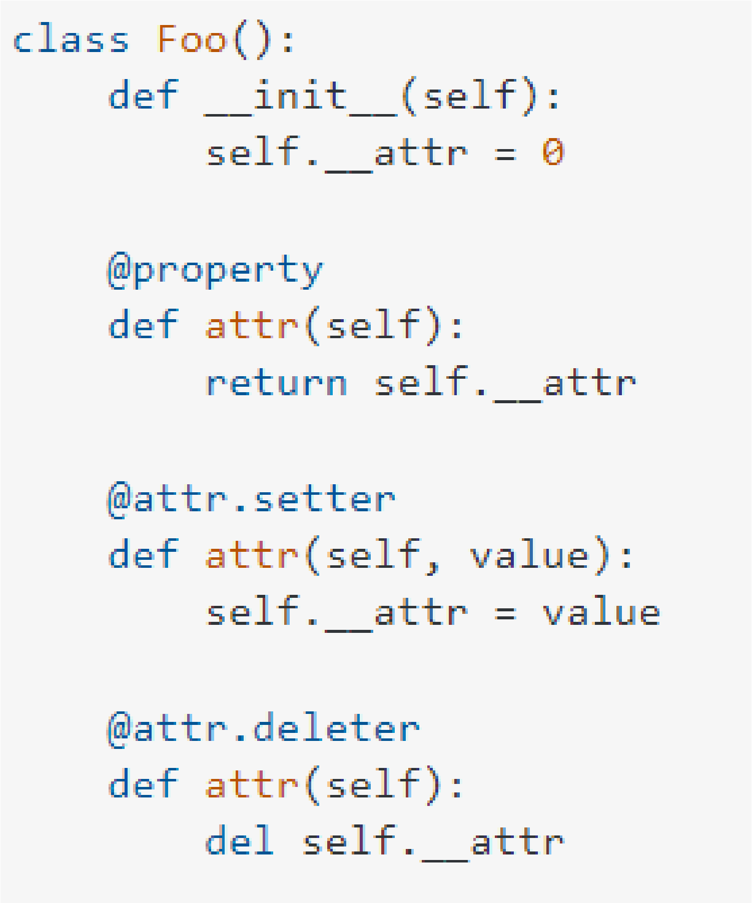
```python
class DocumentStore(object):

    def __init__(capacity):
        self._capacity = capacity
        self._documents = []

    @property
    def capacity(self):
        return self._capacity

    @property
    def documents(self):
        return self._documents

    def add_document(self, document):
        if(len(self._documents) > self._capacity):
            raise Exception('Document store is full')
        self._documents.append(document)

    def __repr__(self):
        return "Document store: " + len(self._documents) +

#To see the output, uncomment the lines belows:
#document_store = DocumentStore(2)
#document_store.add_document("document")
#print(document_store)
```



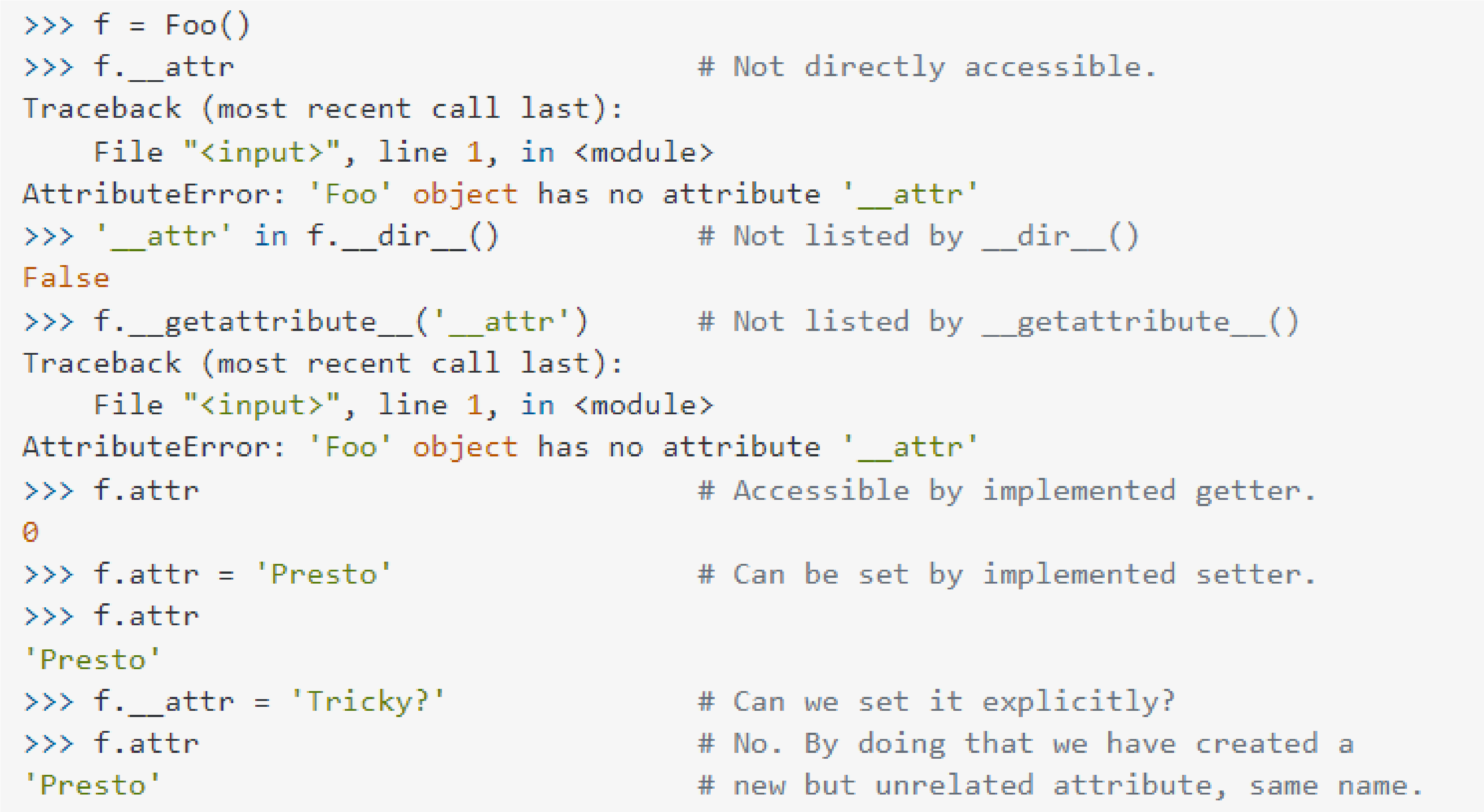
```python
class DocumentStore(object):

    def __init__(self, capacity): ##
        self._capacity = capacity
        self._documents = []

    @property
    def capacity(self):
        return self._capacity

    @property
    def documents(self):
        return self._documents[:] ##

    def add_document(self, document):
        if(len(self._documents) + 1 > self._capacity): ##
            raise Exception('Document store is full')
        self._documents.append(document)

    def __repr__(self):
        return "Document store: " + str(len(self._documents)) + "/" + str(self._capacity) ##

#To see the output, uncomment the lines belows:
#document_store = DocumentStore(2)
#document_store.add_document("document")
#print(document_store)
```

#3

Property decorator



```python
class Foo():
    def __init__(self):
        self.__attr = 0

    @property
    def attr(self):
        return self.__attr

    @attr.setter
    def attr(self, value):
        self.__attr = value

    @attr.deleter
    def attr(self):
        del self.__attr
```



```python
>>> f = Foo()
>>> f.__attr                          # Not directly accessible.
Traceback (most recent call last):
    File "<input>", line 1, in <module>
AttributeError: 'Foo' object has no attribute '__attr'
>>> '__attr' in f.__dir__()           # Not listed by __dir__()
False
>>> f.__getattribute__('__attr')      # Not listed by __getattribute__()
Traceback (most recent call last):
    File "<input>", line 1, in <module>
AttributeError: 'Foo' object has no attribute '__attr'
>>> f.attr                            # Accessible by implemented getter.
0
>>> f.attr = 'Presto'                 # Can be set by implemented setter.
>>> f.attr
'Presto'
>>> f.__attr = 'Tricky?'              # Can we set it explicitly?
>>> f.attr                            # No. By doing that we have created a
'Presto'                              # new but unrelated attribute, same name.
```

https://stackoverflow.com/questions/4555932/public-or-private-attribute-in-python-what-is-the-best-way

#3

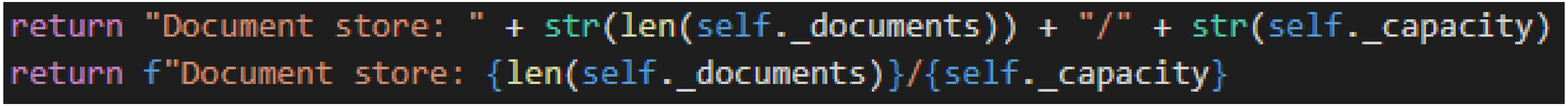
Complexity

	List	Dict	Set
x in S	O(n)	O(1)	O(1)
add	O(1)	O(1)	O(1)
slice, pop(0)	O(n)		
	Array of PyObjects	Hash table	Hash table

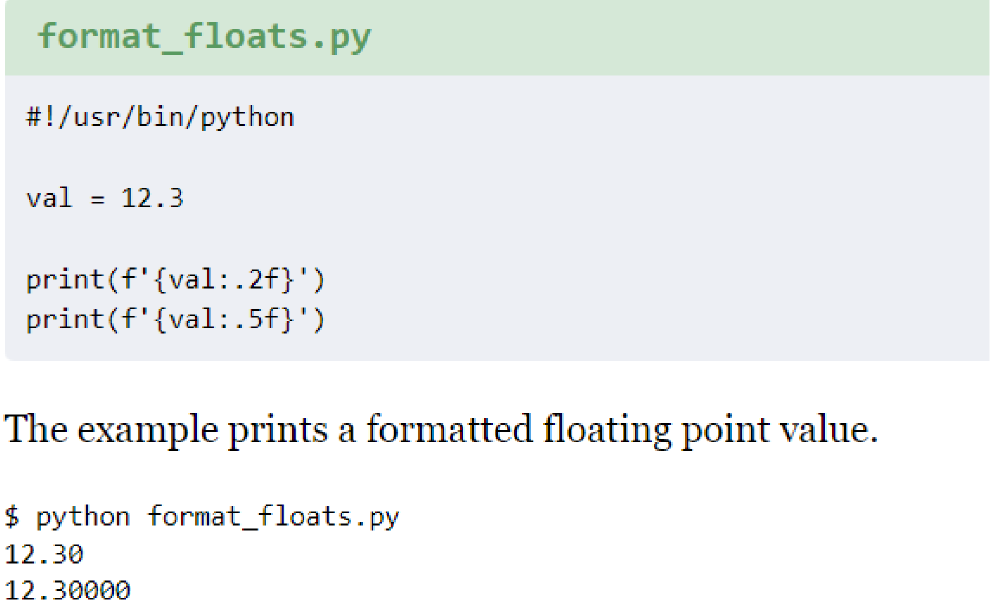
#3

f-string

PEP 498 -- Literal String Interpolation



```
return "Document store: " + str(len(self._documents)) + "/" + str(self._capacity)
return f"Document store: {len(self._documents)}/{self._capacity}
```



format_floats.py

```
#!/usr/bin/python

val = 12.3

print(f'{val:.2f}')
print(f'{val:.5f}')
```

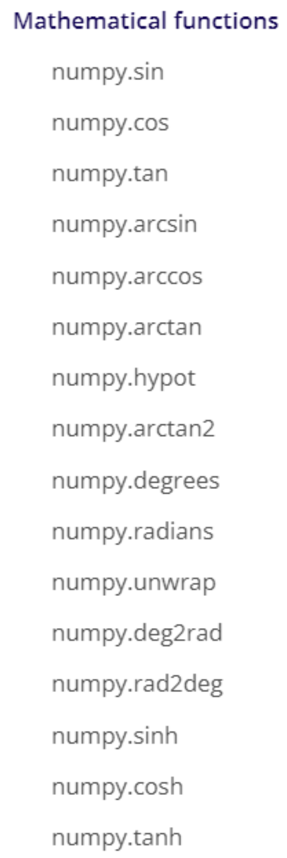
The example prints a formatted floating point value.

```
$ python format_floats.py
12.30
12.30000
```

https://zetcode.com/python/fstring/

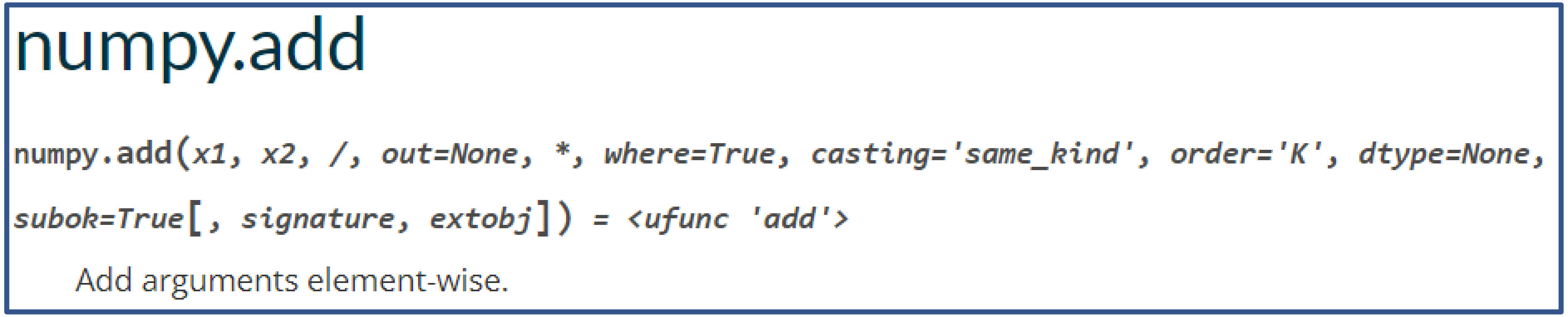
#3

in-place operation



Mathematical functions

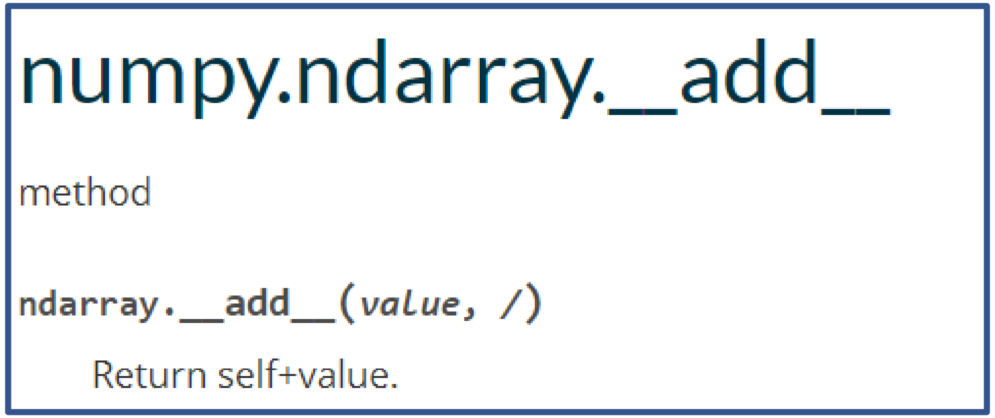
- numpy.sin
- numpy.cos
- numpy.tan
- numpy.arcsin
- numpy.arccos
- numpy.arctan
- numpy.hypot
- numpy.arctan2
- numpy.degrees
- numpy.radians
- numpy.unwrap
- numpy.deg2rad
- numpy.rad2deg
- numpy.sinh
- numpy.cosh
- numpy.tanh



numpy.add

```
numpy.add(x1, x2, /, out=None, *, where=True, casting='same_kind', order='K', dtype=None, subok=True[, signature, extobj]) = <ufunc 'add'>
```

Add arguments element-wise.



numpy.ndarray.__add__

method

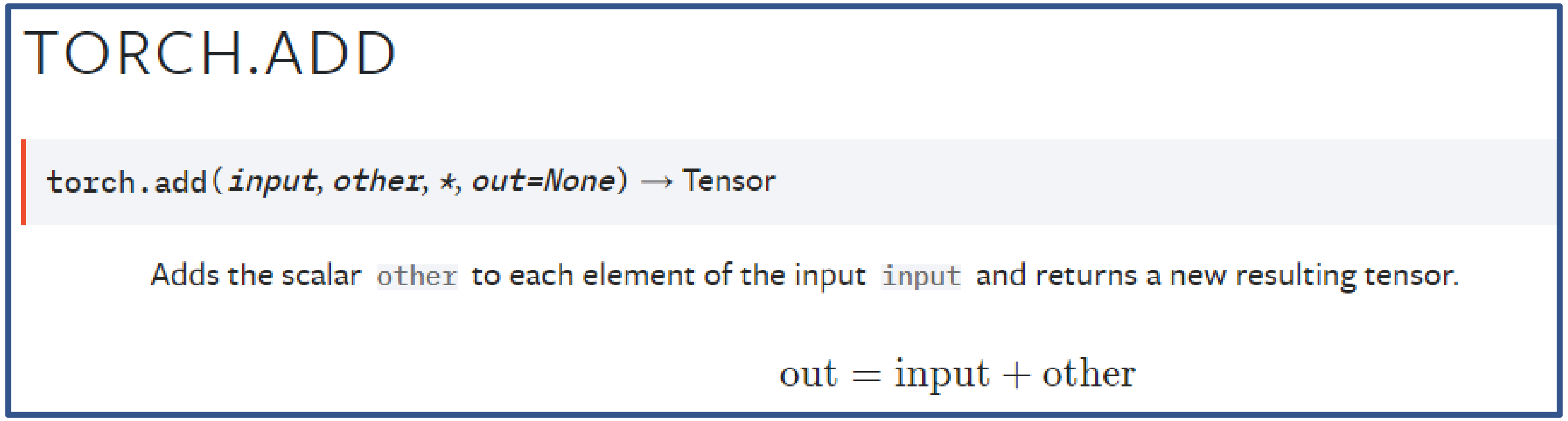
```
ndarray.__add__(value, /)
```

Return self+value.

https://numpy.org/doc/stable/reference/generated/numpy.add.html
https://numpy.org/doc/stable/reference/generated/numpy.ndarray.__add__.html

#3

in-place operation

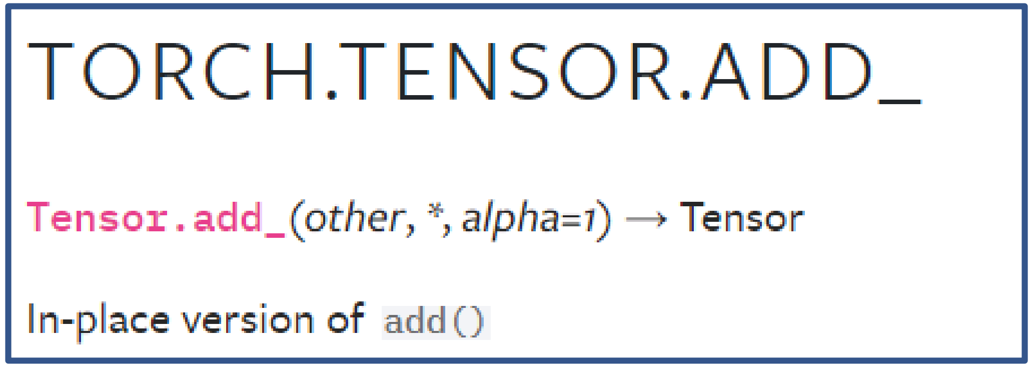


TORCH.ADD

torch.add(*input*, *other*, *, *out=None*) → Tensor

Adds the scalar `other` to each element of the input `input` and returns a new resulting tensor.

$$\text{out} = \text{input} + \text{other}$$



TORCH.TENSOR.ADD_

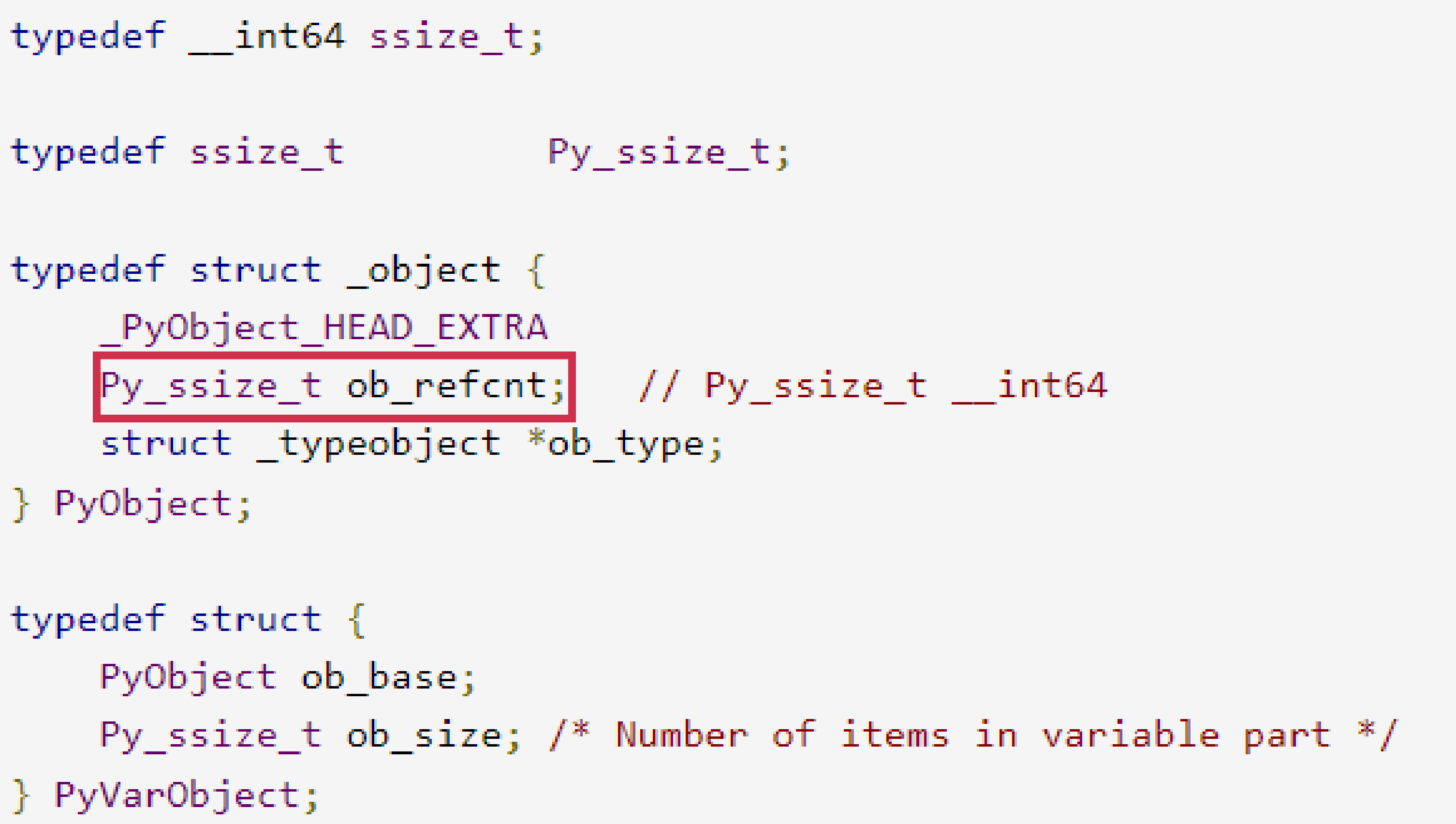
Tensor.add_(*other*, *, *alpha=1*) → Tensor

In-place version of `add()`

https://pytorch.org/docs/stable/generated/torch.add.html
https://pytorch.org/docs/stable/generated/torch.Tensor.add_.html

#3

Python garbage collection

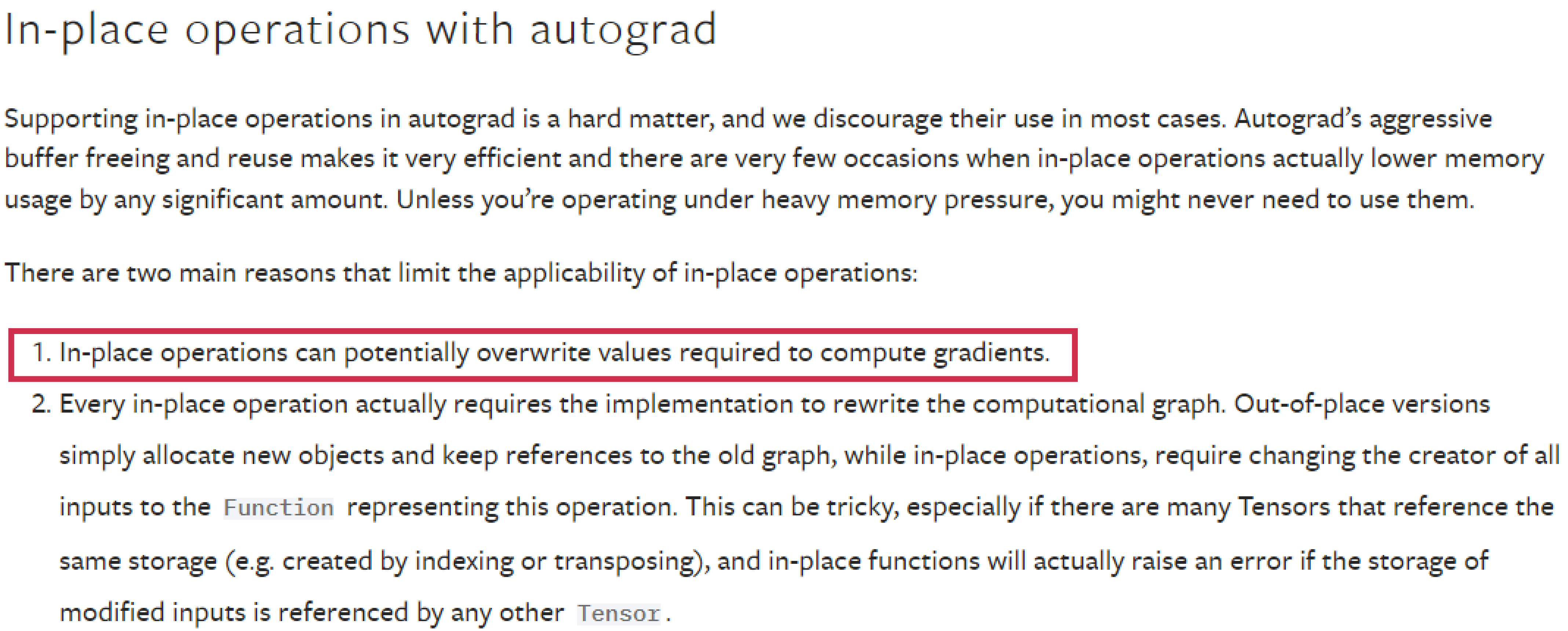


```
typedef __int64 ssize_t;

typedef ssize_t         Py_ssize_t;

typedef struct _object {
    _PyObject_HEAD_EXTRA
    Py_ssize_t ob_refcnt;   // Py_ssize_t __int64
    struct _typeobject *ob_type;
} PyObject;

typedef struct {
    PyObject ob_base;
    Py_ssize_t ob_size; /* Number of items in variable part */
} PyVarObject;
```

https://iter01.com/510827.html

#3

in-place operation



In-place operations with autograd

Supporting in-place operations in autograd is a hard matter, and we discourage their use in most cases. Autograd's aggressive buffer freeing and reuse makes it very efficient and there are very few occasions when in-place operations actually lower memory usage by any significant amount. Unless you're operating under heavy memory pressure, you might never need to use them.

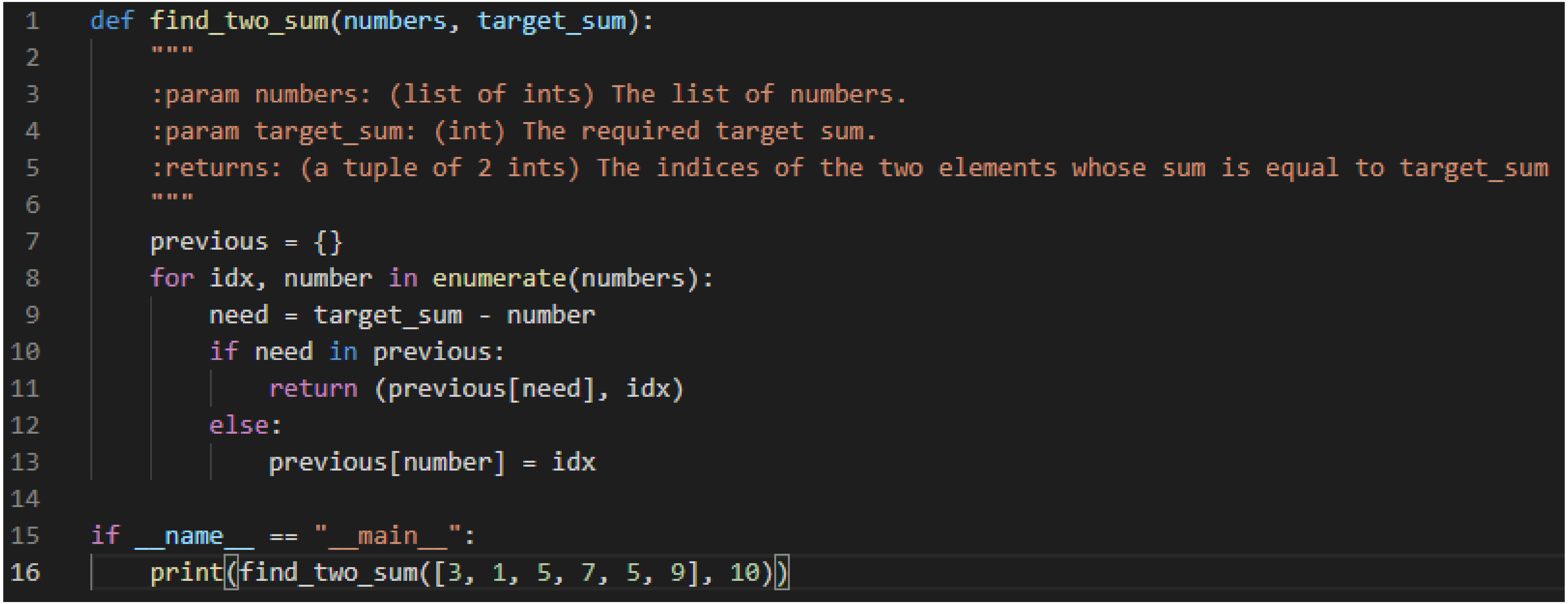
There are two main reasons that limit the applicability of in-place operations:

1. In-place operations can potentially overwrite values required to compute gradients.
2. Every in-place operation actually requires the implementation to rewrite the computational graph. Out-of-place versions simply allocate new objects and keep references to the old graph, while in-place operations, require changing the creator of all inputs to the `Function` representing this operation. This can be tricky, especially if there are many Tensors that reference the same storage (e.g. created by indexing or transposing), and in-place functions will actually raise an error if the storage of modified inputs is referenced by any other `Tensor`.

https://pytorch.org/docs/stable/notes/autograd.html

#4

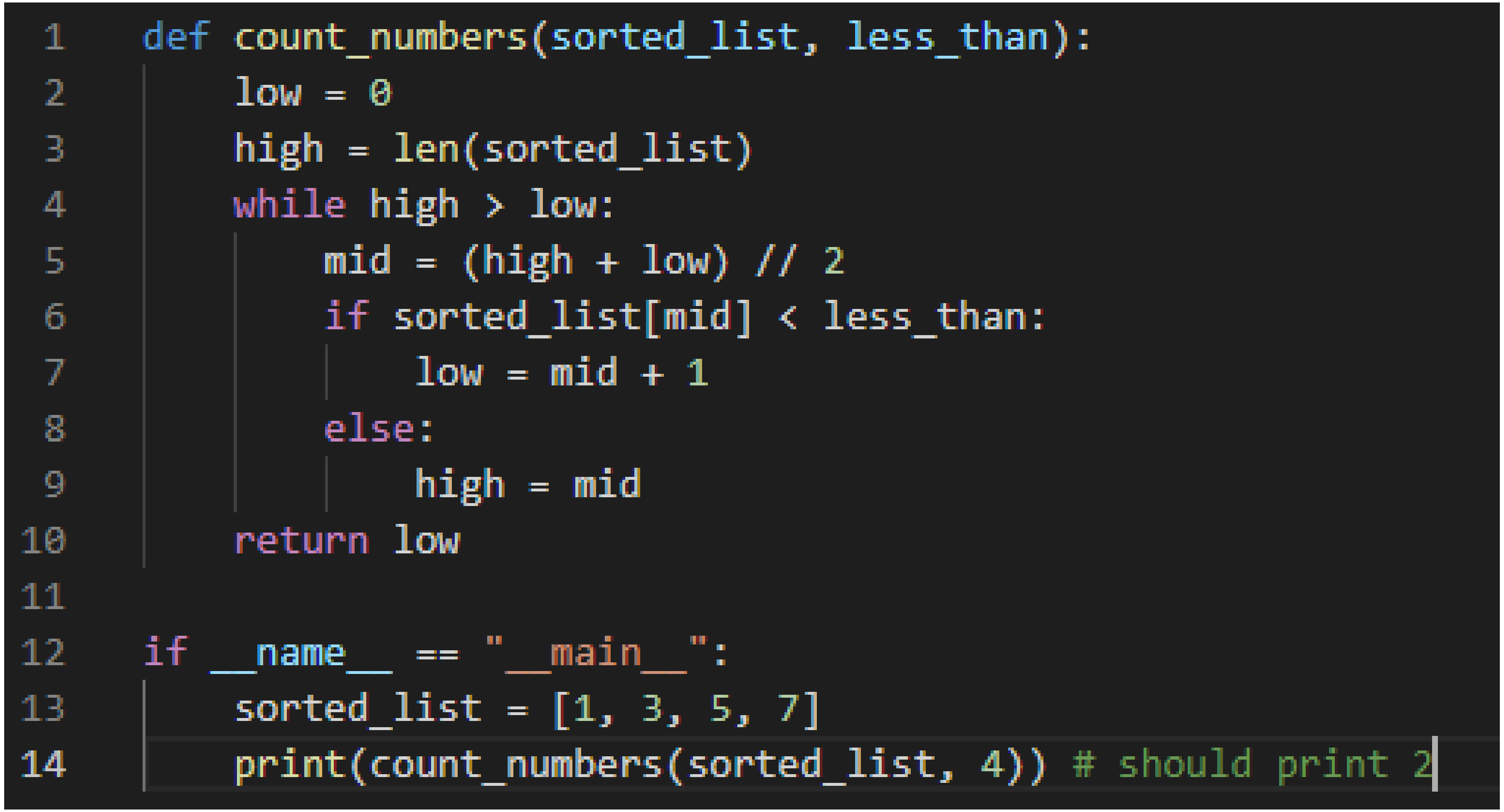
Two Sum



```
def find_two_sum(numbers, target_sum):
    """
    :param numbers: (list of ints) The list of numbers.
    :param target_sum: (int) The required target sum.
    :returns: (a tuple of 2 ints) The indices of the two elements whose sum is equal to target_sum
    """
    previous = {}
    for idx, number in enumerate(numbers):
        need = target_sum - number
        if need in previous:
            return (previous[need], idx)
        else:
            previous[number] = idx

if __name__ == "__main__":
    print(find_two_sum([3, 1, 5, 7, 5, 9], 10))
```

#5

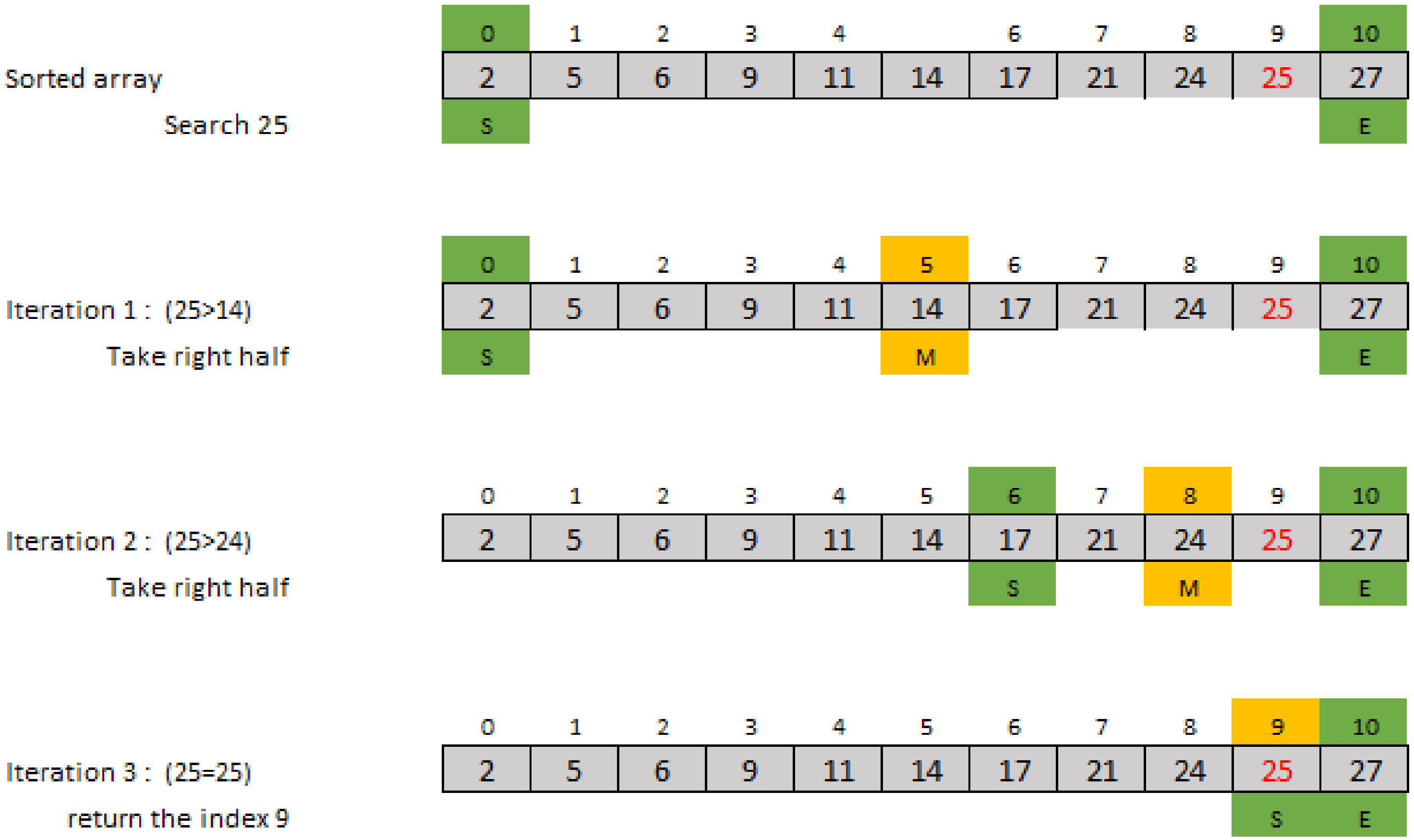
Sorted Search



```python
def count_numbers(sorted_list, less_than):
    low = 0
    high = len(sorted_list)
    while high > low:
        mid = (high + low) // 2
        if sorted_list[mid] < less_than:
            low = mid + 1
        else:
            high = mid
    return low

if __name__ == "__main__":
    sorted_list = [1, 3, 5, 7]
    print(count_numbers(sorted_list, 4)) # should print 2
```

#5

Binary search



Sorted array

Search 25

0	1	2	3	4		6	7	8	9	10
2	5	6	9	11	14	17	21	24	25	27
S										E

Iteration 1 : (25>14)

Take right half

0	1	2	3	4	5	6	7	8	9	10
2	5	6	9	11	14	17	21	24	25	27
S					M					E

Iteration 2 : (25>24)

Take right half

0	1	2	3	4	5	6	7	8	9	10
2	5	6	9	11	14	17	21	24	25	27
						S		M		E

Iteration 3 : (25=25)

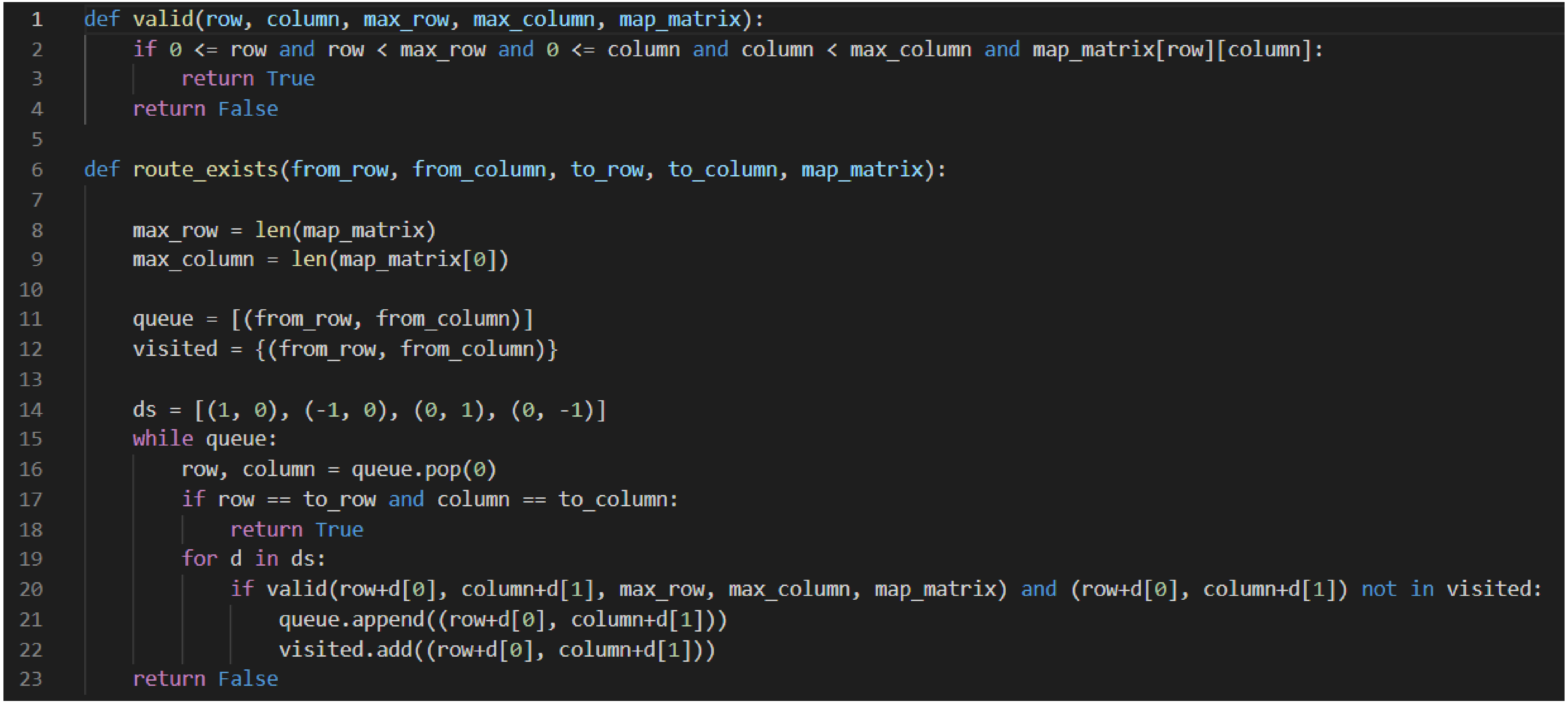
return the index 9

0	1	2	3	4	5	6	7	8	9	10
2	5	6	9	11	14	17	21	24	25	27
									S	E

https://kheri.net/binary-search-algorithm-java-example/

#6

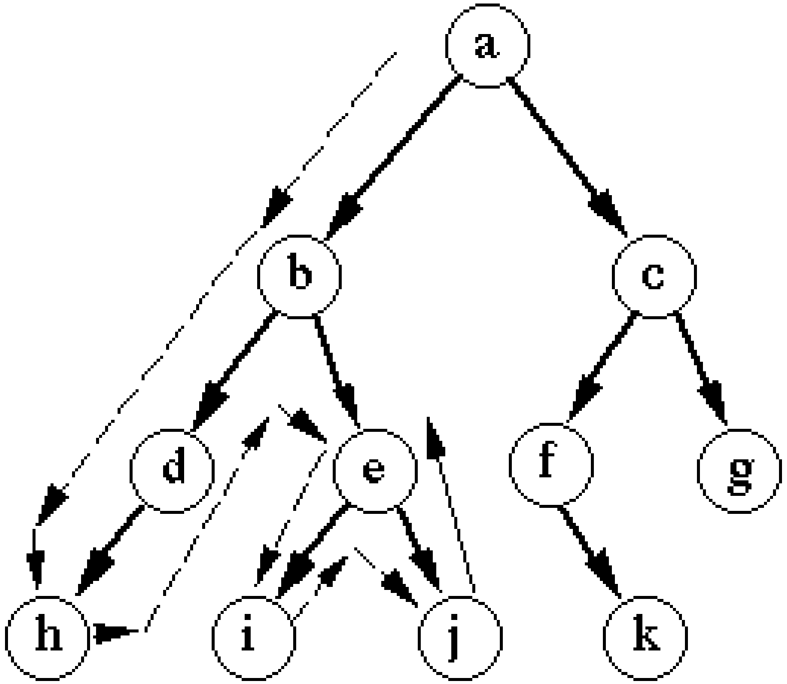
Route Planner



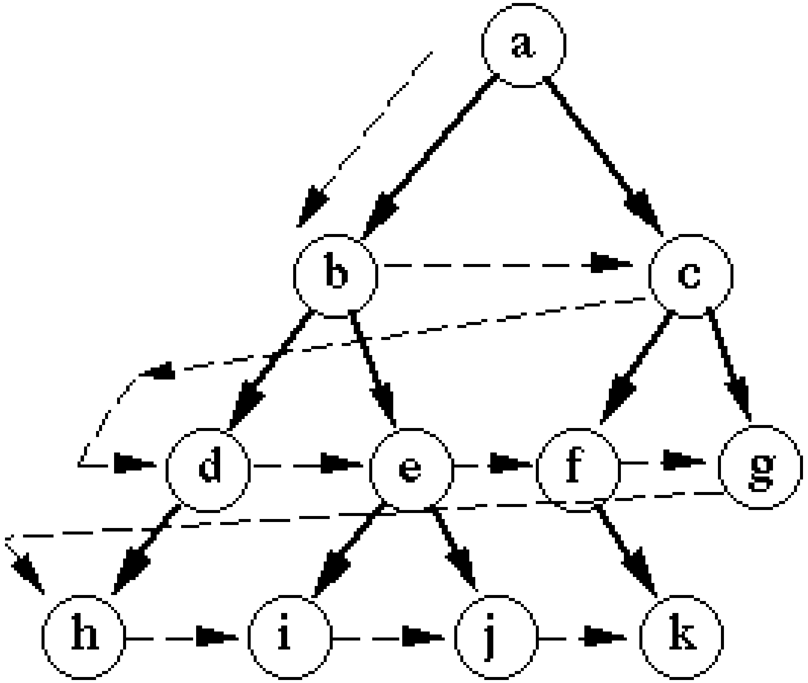
```python
def valid(row, column, max_row, max_column, map_matrix):
    if 0 <= row and row < max_row and 0 <= column and column < max_column and map_matrix[row][column]:
        return True
    return False

def route_exists(from_row, from_column, to_row, to_column, map_matrix):

    max_row = len(map_matrix)
    max_column = len(map_matrix[0])

    queue = [(from_row, from_column)]
    visited = {(from_row, from_column)}

    ds = [(1, 0), (-1, 0), (0, 1), (0, -1)]
    while queue:
        row, column = queue.pop(0)
        if row == to_row and column == to_column:
            return True
        for d in ds:
            if valid(row+d[0], column+d[1], max_row, max_column, map_matrix) and (row+d[0], column+d[1]) not in visited:
                queue.append((row+d[0], column+d[1]))
                visited.add((row+d[0], column+d[1]))
    return False
```

#6

BFS/DFS



Depth-first search



Breadth-first search

http://www.cse.unsw.edu.au/~billw/Justsearch.html

Others

Recommendation

OS	Linux
Environment	Anaconda (python >= 3.8) + PyTorch >= 1.9.0
IDE	VS Code

GPU is required to expedite your deep learning projects

PLEASE start to learn PyTorch as early as possible!

Others

If you are already a PyTorch expert

TENSORS
JIT
PRODUCTION
Q&A

PyTorch under the hood

A guide to understand PyTorch internals

Christian S. Perone
(*christian.perone@gmail.com*)
http://blog.christianperone.com

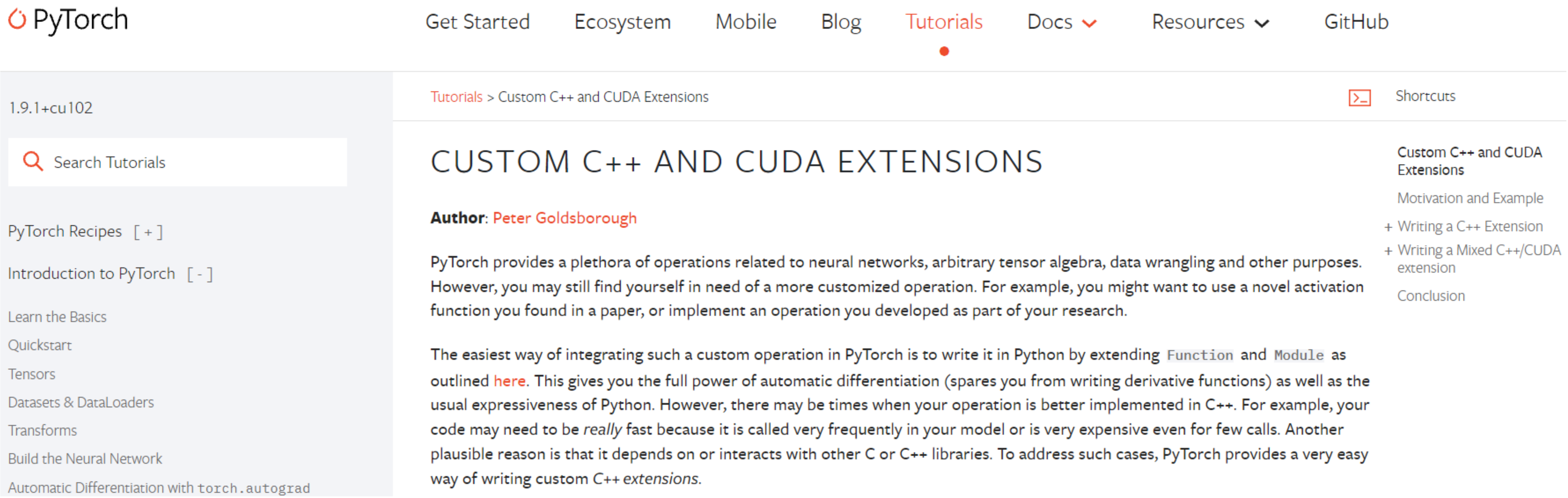
PyData Montreal, Feb 2019

PyTorch under the hood - Christian S. Perone (2019)

https://speakerdeck.com/perone/pytorch-under-the-hood

Others

If you are already a PyTorch expert



CUSTOM C++ AND CUDA EXTENSIONS

Author: Peter Goldsborough

PyTorch provides a plethora of operations related to neural networks, arbitrary tensor algebra, data wrangling and other purposes. However, you may still find yourself in need of a more customized operation. For example, you might want to use a novel activation function you found in a paper, or implement an operation you developed as part of your research.

The easiest way of integrating such a custom operation in PyTorch is to write it in Python by extending `Function` and `Module` as outlined here. This gives you the full power of automatic differentiation (spares you from writing derivative functions) as well as the usual expressiveness of Python. However, there may be times when your operation is better implemented in C++. For example, your code may need to be *really* fast because it is called very frequently in your model or is very expensive even for few calls. Another plausible reason is that it depends on or interacts with other C or C++ libraries. To address such cases, PyTorch provides a very easy way of writing custom *C++ extensions*.

“The rest of this note will walk through a practical example of writing and using a C++ (and CUDA) extension. If you are being chased or someone will fire you if you don’ t get that op done by the end of the day, you can skip this section and head straight to the implementation details in the next section.”

https://pytorch.org/tutorials/advanced/cpp_extension.html

Thank you