

# The Standard Library Tour

**Mia Bajic**

## About me



- software engineer at Ataccama

## About me



- software engineer at Ataccama
- over 5 years of diverse experience in the IT industry, ranging from tech support and testing to analysis and development

## About me



- software engineer at Ataccama
- over 5 years of diverse experience in the IT industry, ranging from tech support and testing to analysis and development
- based in Prague

## About me



- software engineer at Ataccama
- over 5 years of diverse experience in the IT industry, ranging from tech support and testing to analysis and development
- based in Prague
- co-organizer of Pyvo - Prague Python meetups & co-organizer of PyCon CZ

What is the standard library?

What is the standard library?

- a collection of modules and functions included with Python

## What is the standard library?

- a collection of modules and functions included with Python
- offers numerous functionalities without installing 3rd party libraries



## What is the standard library?

- a collection of modules and functions included with Python
- offers numerous functionalities without installing 3rd party libraries
- functionalities include interacting with OS, running servers, scientific computing, debugging, data manipulation, and more

Why should you use the standard library?

Why should you use the standard library?

- you're not re-inventing the wheel when it comes to finding solutions

## Why should you use the standard library?

- you're not re-inventing the wheel when it comes to finding solutions
- the solutions that are available have already been optimized for efficiency

## Why should you use the standard library?

- you're not re-inventing the wheel when it comes to finding solutions
- the solutions that are available have already been optimized for efficiency
- using these pre-existing solutions can help avoid encountering bugs that have already been fixed

What is this talk about?

What is this talk about?

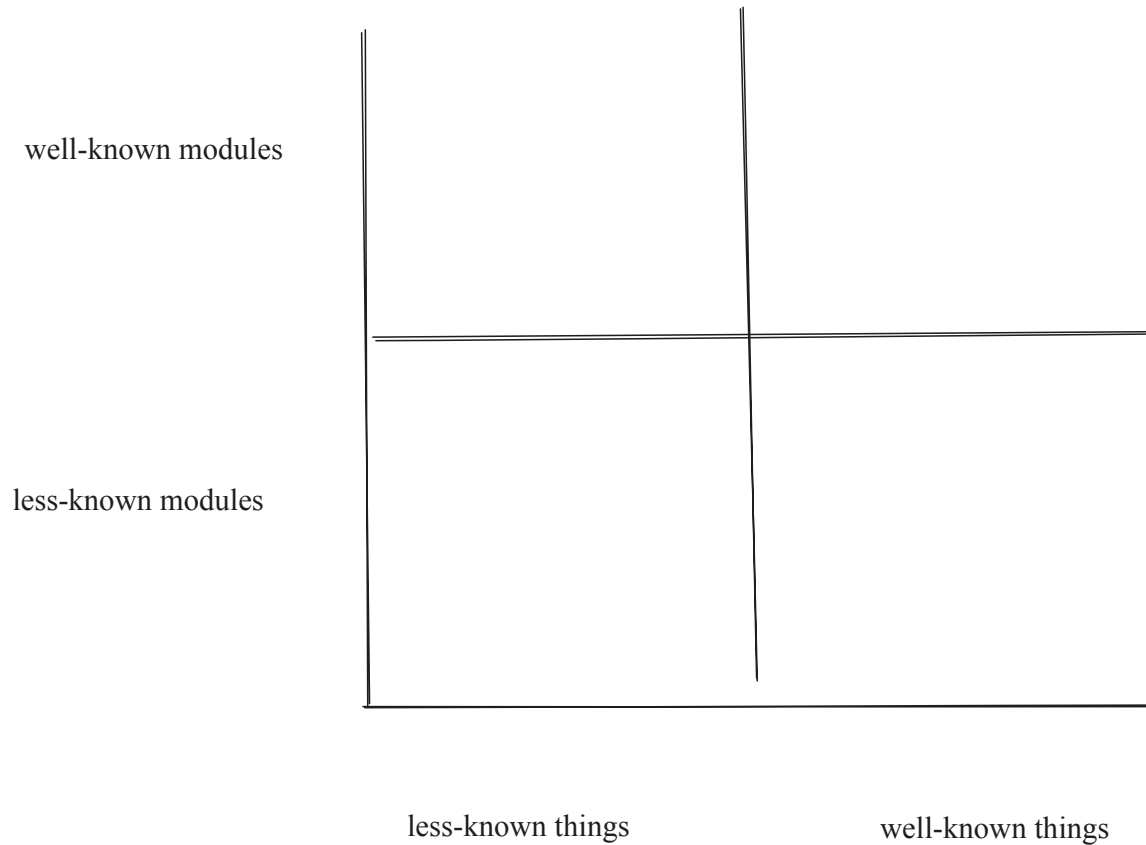
- a brief overview of lesser-known features of the standard library

What is this talk about?

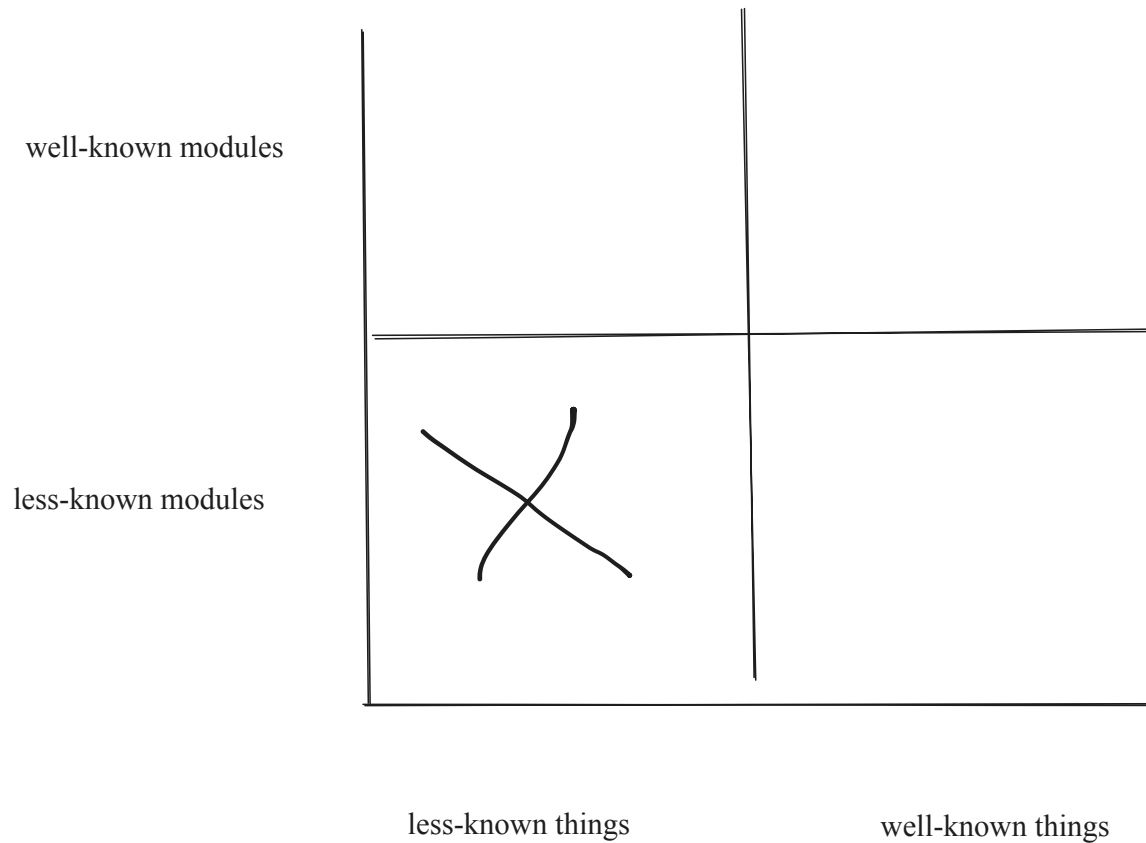
- a brief overview of lesser-known features of the standard library
- the aim is to discover the unknown unknowns - features that you didn't even know exist



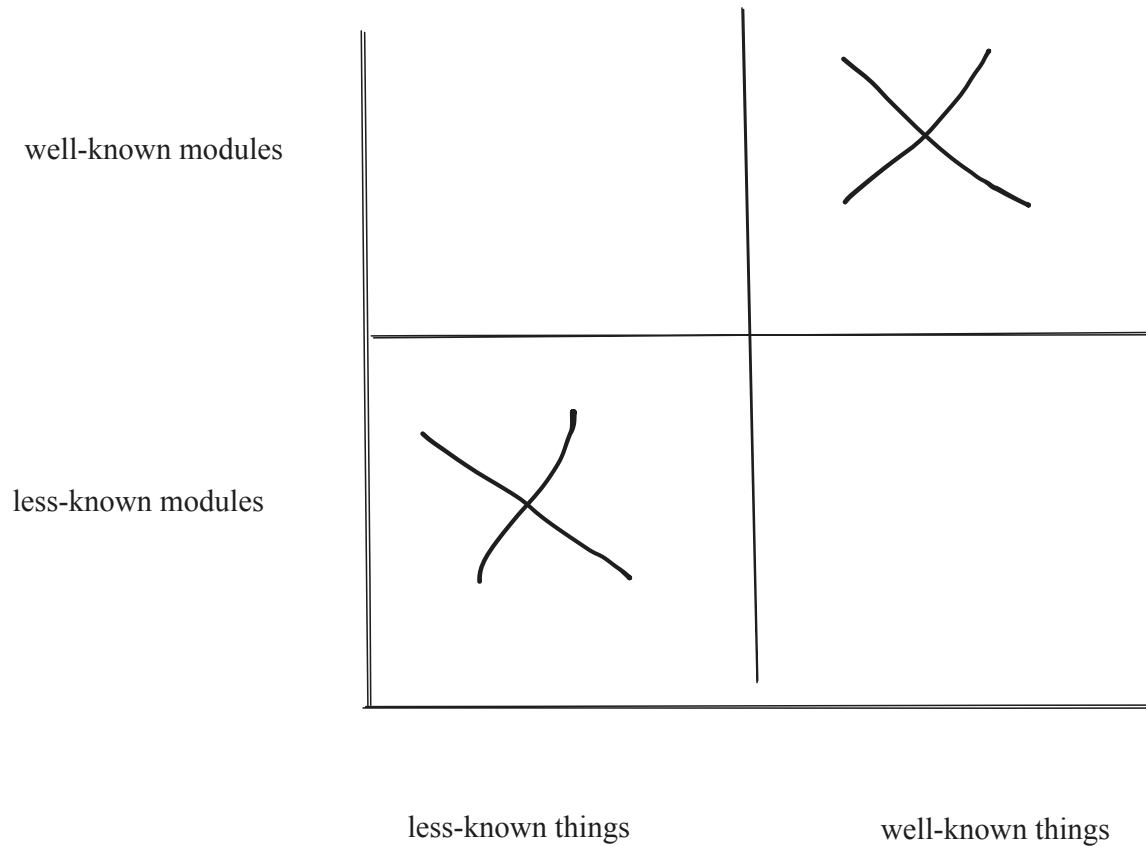
# What is this talk about?



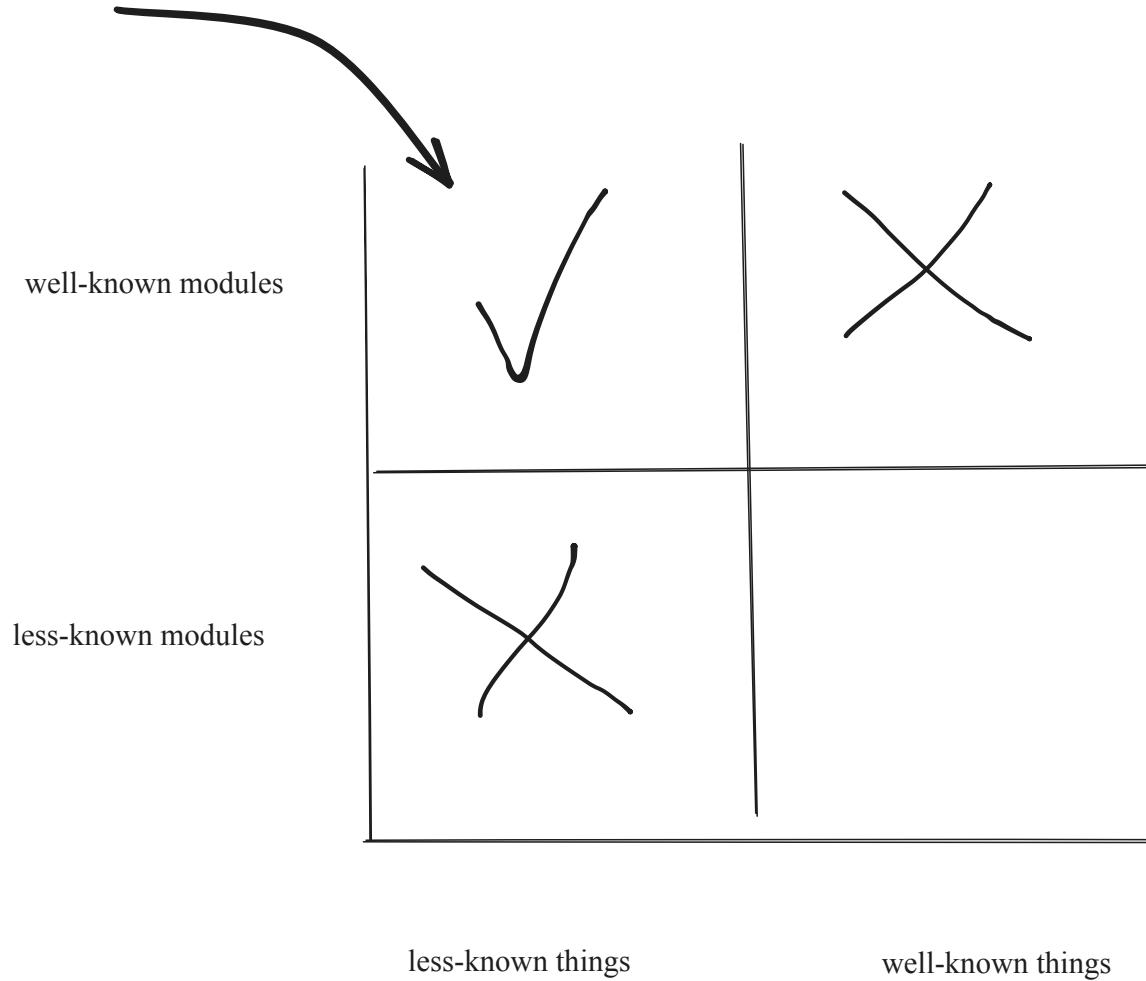
# What is this talk about?



# What is this talk about?



# Well-known libraries that do less-known things



## Disclaimer

- all code examples are illustrative

# Functools

# Functools

- one of the most frequently used modules inside the standard library

# Functools

- one of the most frequently used modules inside the standard library
- functools includes a variety of tools for working with functions, including tools for modifying function behavior or creating function-like objects



# Functools

```
1 def add(x, y):
2     if isinstance(x, int) and isinstance(y, int):
3         return x + y
4     elif isinstance(x, str) and isinstance(y, str):
5         return x + " " + y
6     else:
7         raise ValueError("Unsupported data type")
```

## Functools

```
1 def add(x, y):
2     if isinstance(x, int) and isinstance(y, int):
3         return x + y
4     elif isinstance(x, str) and isinstance(y, str):
5         return x + " " + y
6     else:
7         raise ValueError("Unsupported data type")
8
9 print(add(1, 2))
10 print(add("Hello", "World!"))
11 print(add(1, "World!"))
```

# Functools

```
1 3
2 Hello World!
3 Traceback (most recent call last):
4   File
    "/home/mia/Documents/repos/osobní/python/demo/europython/si
ngledispatch.py", line 12, in <module>
5     print(add(1, "World!"))
6   File
    "/home/mia/Documents/repos/osobní/python/demo/europython/si
ngledispatch.py", line 7, in add
7     raise ValueError("Unsupported data type")
8 ValueError: Unsupported data type
```

# Functools

```
1 from functools import singledispatch
2
3
4 @singledispatch
5 def add(x, y):
6     raise ValueError("Unsupported data type")
7
8
9 @add.register
10 def _(x: int, y: int):
11     return x + y
12
13
14 @add.register
15 def _(x: str, y: str):
16     return x + " " + y
```

## Functools

- @singledispatch is used for function overloading
  - creating several methods with the same name which differ from each other in the type of input parameters or the number of input parameters

# Functools

- @singledispatch is used for function overloading
  - creating several methods with the same name which differ from each other in the type of input parameters or the number of input parameters
- commonly used for cases when you work with different data types as input to your functions

## Functools

- the advantage of `@singledispatch` over `if/elif/else` type checking:

## Functools

- the advantage of @singledispatch over if/elif/else type checking:
  - easier to modify - each function that handles one type is independent and can be modified independently of others



## Functools

- the advantage of @singledispatch over if/elif/else type checking:
  - easier to modify - each function that handles one type is independent and can be modified independently of others
  - the code is cleaner and more readable

## Functools

- `@singledispatch` dispatches only for the first argument

## Functools

- @singledispatch dispatches only for the first argument
- the downside is, if there are multiple arguments, third-party libraries need to be used

# Functools

# Functools

```
1 def add_to_two(x, y=2):  
2     return x + y  
3  
4 print(add_to_two(3))
```

# Functools

```
1 def add_to_three(x, y=3):  
2     return x + y  
3  
4 print(add_to_three(2))
```

# Functools

```
1 import functools
2
3
4 def add(x, y):
5     return x + y
6
7 add_to_two = functools.partial(add, y=2)
8 add_to_three = functools.partial(add, y=3)
9
10 print(add_to_two(3))
11 print(add_to_three(3))
```

# Functools

```
1 import functools
2
3
4 def add(x, y):
5     return x + y
6
7 add_to_two = functools.partial(add, y=2)
8 add_to_three = functools.partial(add, y=3)
9
10 print(add_to_two(3))
11 print(add_to_three(3))
12
13 5
14 6
```



## Functools

- `partial` is used to create a new function with some of the arguments of the original function

## Functools

- `partial` is used to create a new function with some of the arguments of the original function
- it can be used with any callable, including built-in functions, methods from other libraries, args and kwargs

## Functools

- the main advantage of using partial is code reusability and adhering to the DRY principle (Don't Repeat Yourself)

## Functools

- the main advantage of using partial is code reusability and adhering to the DRY principle (Don't Repeat Yourself)
- a common use case is when you need to call a function with the same argument multiple times

## Functools

- the main advantage of using partial is code reusability and adhering to the DRY principle (Don't Repeat Yourself)
- a common use case is when you need to call a function with the same argument multiple times
- the main downside is that partial may not be intuitive for new Python developers

# Functools

```
1 import functools
2
3 def fibonacci(n):
4     if n < 2:
5         return n
6     return fibonacci(n-1) + fibonacci(n-2)
```

# Functools

```
1 import functools
2
3 @functools.lru_cache(maxsize=None)
4 def fibonacci(n):
5     if n < 2:
6         return n
7     return fibonacci(n-1) + fibonacci(n-2)
```

# Functools

```
1 import functools
2
3 @functools.cache
4 def fibonacci(n):
5     if n < 2:
6         return n
7     return fibonacci(n-1) + fibonacci(n-2)
```



## Functools

- `@lru_cache` (Least Recently Used) stores the results of function calls

## Functools

- `@lru_cache` (Least Recently Used) stores the results of function calls
- `@lru_cache` checks for the key in cache dictionary, when the key is present the wrapper returns the value and updates the cache hit info

## Functools

- @lru\_cache (Least Recently Used) stores the results of function calls
- it checks for the key in cache dictionary, when the key is present the wrapper returns the value and updates the cache hit info
- if the key is missing, the wrapper calls the user function with passed arguments, updates the cache miss info and returns the result

## Functools

- if the cache is full, the `@lru_cache` evicts the old items and adds new ones

# Functools

- @cache is available from version 3.9

## Functools

- @cache is available from version 3.9
- it is same as @lru\_cache(maxsize=None)

## Functools

- @cache is available from version 3.9
- it is same as @lru\_cache(maxsize=None)
- @cache doesn't evict the old values, so it's faster

# Functools

```
1 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 -mtimeit -s 'import fibonacci_without_cache as f'  
    'f.fibonacci(n=20)'  
2 200 loops, best of 5: 1.03 msec per loop  
3  
4 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 -mtimeit -s 'import fibonacci_with_lru_cache as f'  
    'f.fibonacci(n=20)'  
5 5000000 loops, best of 5: 97.1 nsec per loop  
6  
7 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 -mtimeit -s 'import fibonacci_with_cache as f'  
    'f.fibonacci(n=20)'  
8 2000000 loops, best of 5: 97.8 nsec per loop
```



Itertools

## Itertools

- provide various functions that create iterators for efficient looping

## Itertools

- provide various functions that create iterators for efficient looping
- they are useful for handling large data streams

## Itertools

```
1 list1 = [1, 2, 3]
2 list2 = [4, 5, 6]
3
4 product = []
5
6 for i in list1:
7     for j in list2:
8         product.append((i, j))
9 print(product)
10
11 mia@mias-
   ntb:~/Documents/repos/osobní/python/demo/europython$
   python3 product.py
12 [(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4),
    (3, 5), (3, 6)]
```

# Itertools

```
1 import itertools
2
3 list1 = [1, 2, 3]
4 list2 = [4, 5, 6]
5
6 print(list(itertools.product(list1, list2)))
7
8 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 product.py
9 [(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4),
  (3, 5), (3, 6)]
```

## Itertools

- `product()` returns the cartesian product of two iterables

## Itertools

```
1 list1 = [1, 2, 3, 4]
2 list2 = [4, 5, 6]
3
4 print(list(filter(lambda i: i in list1, list2)))
5
6 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 filter.py
7 [4]
```

# Itertools

```
1 import itertools
2
3
4 list1 = [1, 2, 3, 4]
5 list2 = [4, 5, 6]
6
7 print(list(itertools.filterfalse(lambda i: i in list1,
8 list2)))
9 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 filter.py
10 [5, 6]
```



## Itertools

- `filterfalse()` filters elements from an iterable returning only those for which the predicate is false

## Itertools

- `filterfalse()` filters elements from an iterable returning only those for which the predicate is false
- the opposite of built-in `filter()`

## Itertools

```
1 numbers = [1, 2, 3]
2 letters = ['a', 'b', 'c', 'd', 'e']
3
4 print(list(zip(numbers, letters)))
```

## Itertools

```
1 numbers = [1, 2, 3]
2 letters = ['a', 'b', 'c', 'd', 'e']
3
4 print(list(zip(numbers, letters)))
5
6 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 zip.py
7 [(1, 'a'), (2, 'b'), (3, 'c')]
```

## Itertools

```
1 import itertools
2
3 numbers = [1, 2, 3]
4 letters = ['a', 'b', 'c', 'd', 'e']
5
6 print(list(itertools.zip_longest(numbers, letters)))
7
8 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 zip_longest.py
9 [(1, 'a'), (2, 'b'), (3, 'c'), (None, 'd'), (None, 'e')]
```

# Collections

## Collections

```
1 dict1 = {"a": 1, "b": 2, "c": 3}
2 dict2 = {"d": 4, "e": 5, "f": 6}
```

## Collections

```
1 dict1 = {"a": 1, "b": 2, "c": 3}
2 dict2 = {"d": 4, "e": 5, "f": 6}
3
4 if "c" in dict1:
5     print(dict1["c"])
6 elif "c" in dict2:
7     print(dict2["c"])
8 else:
9     print("Not found")
10
11 mia@mias-
   ntbt~/Documents/repos/osobní/python/demo/europython$
   python3 chainmap.py
12 3
```



# Collections

```
1 from collections import ChainMap
2
3 dict1 = {"a": 1, "b": 2, "c": 3}
4 dict2 = {"d": 4, "e": 5, "f": 6}
5
6 chain_dict = ChainMap(dict1, dict2)
7 print(chain_dict["c"])
8
9 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 chainmap.py
10 3
```

## Collections

```
1 dict1 = {"a": 1, "b": 2, "c": 3}
2 dict2 = {"d": 4, "e": 5, "f": 6}
3
4 new_dict = {}
5 new_dict.update(dict1)
6 new_dict.update(dict2)
7
8 print(new_dict)
9
10 mia@mias-
   ntb:~/Documents/repos/osobní/python/demo/europython$
   python3 chainmap.py
11 {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6}
```

## Collections

- ChainMap references already existing dictionaries and doesn't copy any data

## Collections

- ChainMap references already existing dictionaries and doesn't copy any data
- it groups multiple dictionaries into one and provides a single, dynamic view

## Collections

- ChainMap references already existing dictionaries and doesn't copy any data
- it groups multiple dictionaries into one and provides a single, dynamic view
- when one of the dictionaries gets updated, the update is visible in ChainMap as well

## Collections

```
1 dict1 = {"a": 1, "b": 2, "c": 3}
2
3 print(dict1["d"])
```

# Collections

```
1 dict1 = {"a": 1, "b": 2, "c": 3}
2
3 print(dict1["d"])
4
5 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 default_dict.py
6 Traceback (most recent call last):
7   File
  "/home/mia/Documents/repos/osobní/python/demo/europython/de
  fault_dict.py", line 3, in <module>
8     print(dict1["d"])
9 KeyError: 'd'
```

## Collections

```
1 from collections import defaultdict
2
3 dict1 = defaultdict(lambda: None, {"a": 1, "b": 2, "c": 3})
4
5 print(dict1["d"])
```



## Collections

```
1 from collections import defaultdict
2
3 dict1 = defaultdict(lambda: None, {"a": 1, "b": 2, "c": 3})
4
5 print(dict1["d"])
6
7 mia@mias-
  ntbtb:~/Documents/repos/osobn1/python/demo/europython$
  python3 default_dict.py
8 None
```

## Collections

- `defaultdict` is a container-like dictionary that returns a default value for a non-existing key

## Collections

- defaultdict is a container-like dictionary that returns a default value for a non-existing key
- it's commonly used for grouping or counting elements in a collection

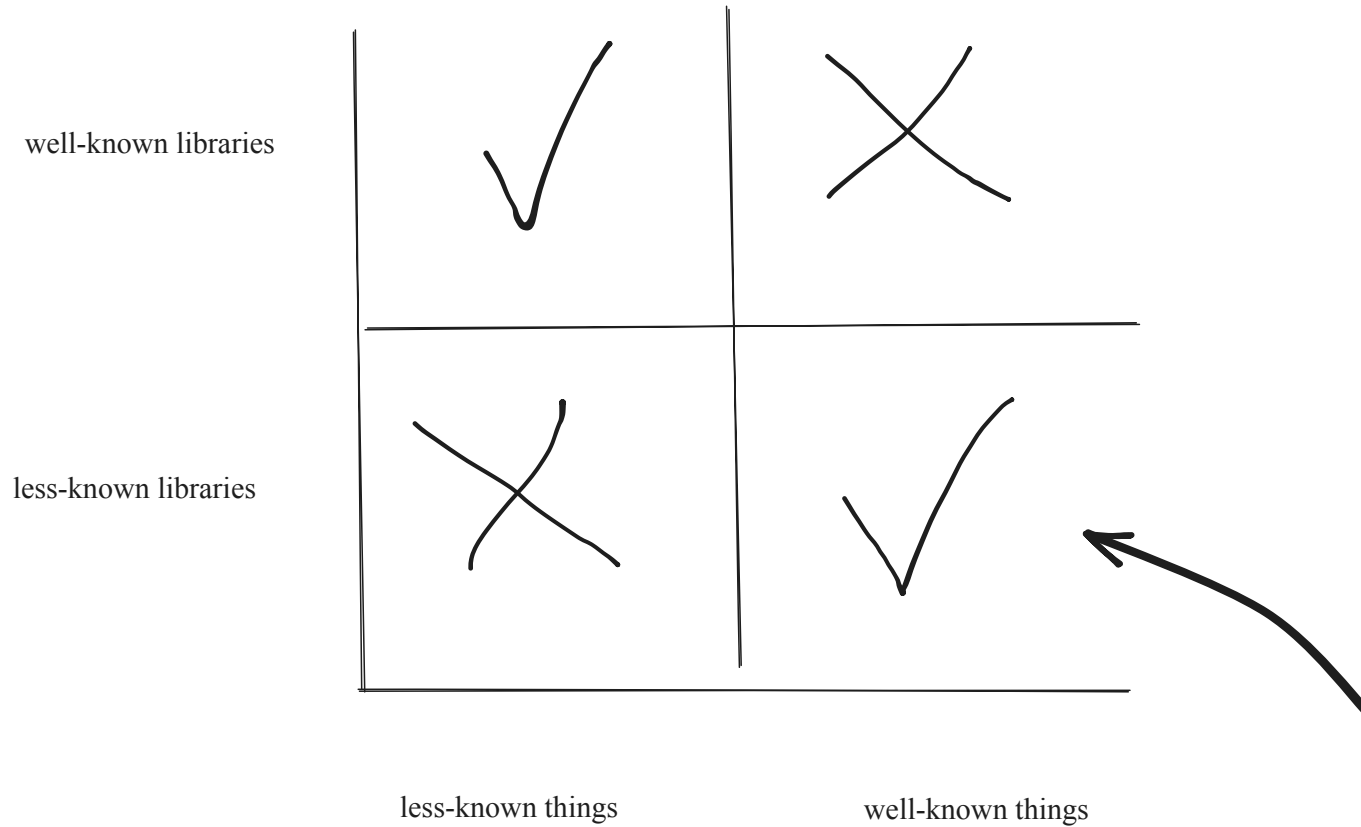
# Collections

- defaultdict is a container-like dictionary that returns a default value for a non-existing key
- it's commonly used for grouping or counting elements in a collection
- defaultdict can make code simpler and more readable by eliminating the need to check if a key is already present in the dictionary before doing operations on it

## Collections

```
1 dict2 = defaultdict(lambda: None, defaultdict(lambda: None,  
    {"a": 1, "b": 2, "c": 3}))  
2  
3 print(dict2["d"])  
4  
5 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 default_dict.py  
6 None
```

# Less-known modules which do well-known things



Testing

# Testing

```
1 def add(x, y):  
2     return x + y
```



# Testing

```
1 def add(x, y):
2     """
3     Adds the two input numbers.
4
5     >>> add(2, 3)
6     5
7     >>> add(-1, 1)
8     0
9     >>> add(-10, -5)
10    -15
11    """
12    return x + y
13
14 if __name__ == "__main__":
15     import doctest
16     doctest.testmod()
```

# Testing

```
1 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 testmod.py  
2
```

# Testing

```
1 def add(x, y):
2     """
3     Adds the two input numbers.
4
5     >>> add(2, 3)
6     5
7     >>> add(-1, 1)
8     0
9     >>> add(-10, -5)
10    15 # this line changed
11    """
12    return x + y
13
14 if __name__ == "__main__":
15     import doctest
16     doctest.testmod()
```

## Example

```
1 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 testmod.py  
2 *****  
  *****  
3 File  
  "/home/mia/Documents/repos/osobní/python/demo/europython/t  
  estmod.py", line 9, in __main__.add  
4 Failed example:  
5     add(-10, -5)  
6 Expected:  
7     15  
8 Got:  
9     -15  
10 *****  
    *****  
11 1 items had failures:
```

# Testing

- Testmod() is used for simple scenarios

# Testing

- Testmod() is used for simple scenarios
- mostly for quick-and-dirty kind of testing and documenting simple scenarios

# Testing

- Testmod() is used for simple scenarios
- mostly for quick-and-dirty kind of testing and documenting simple scenarios
- test cases are readable to humans - it allows you to test and document your code in the same time

# Comparing sequences



## Comparing sequences

```
1 string1 = "Hello world!"  
2 string2 = "Hello World!"
```

## Comparing sequences

```
1 import difflib
2 from pprint import pprint
3
4 string1 = "Hello world!"
5 string2 = "Hello World!"
6
7 d = difflib.Differ()
8 result = list(d.compare(string1, string2))
9
10 pprint(result)
```

# Comparing sequences

```
1 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 differ.py  
2  ['  H',  
3   '  e',  
4   '  l',  
5   '  l',  
6   '  o',  
7   '   ',  
8   '- w',  
9   '+ W',  
10  '  o',  
11  '  r',  
12  '  l',  
13  '  d',  
14  '  !']
```

## Comparing sequences

```
1 import difflib
2 from pprint import pprint
3
4 string1 = "Hello world!"
5 string2 = "Hello World!"
6
7 s = difflib.SequenceMatcher(None, string1, string2)
8 print(s.ratio())
```

## Comparing sequences

```
1 import difflib
2 from pprint import pprint
3
4 string1 = "Hello world!"
5 string2 = "Hello World!"
6
7 s = difflib.SequenceMatcher(None, string1, string2)
8 print(s.ratio())
9
10 mia@mias-
   ntb:~/Documents/repos/osobní/python/demo/europython$
   python3 differ.py
11 0.9166666666666666
```

## Comparing sequences

- DiffLib is used for comparing pair of sequences of any type (strings, tuples, lists etc.) as long as the sequence elements are hashable

## Comparing sequences

- DiffLib is used for comparing pair of sequences of any type (strings, tuples, lists etc.) as long as the sequence elements are hashable
- uses the Ratcliff/Obershelp algorithm
  - number of overlapping characters between the two strings \* 2 / total number of characters in both strings

## Comparing sequences

- with `DiffLib.ratio()` we can measure the similarity of the sequences



## Comparing sequences

- with `DiffLib.ratio()` we can measure the similarity of the sequences
  - values between 0 (no match) and 1 (identical match)

## Comparing sequences

- with `DiffLib.ratio()` we can measure the similarity of the sequences
  - values between 0 (no match) and 1 (identical match)
  - a rule of thumb is `ratio()` value over 0.6 means the sequences are close matches

# Comparing files and directories

## Comparing files and directories

```
1 import filecmp
2
3 file1 = "./dir1/file1.py"
4 file2 = "./dir2/file2.py"
5
6 cmp = filecmp.cmp(file1, file2)
7
8 print(cmp)
```

## Comparing files and directories

```
1 import filecmp
2
3 file1 = "./dir1/file1.py"
4 file2 = "./dir2/file2.py"
5
6 cmp = filecmp.cmp(file1, file2)
7
8 print(cmp)
9
10 mia@mias-
   ntb:~/Documents/repos/osobní/python/demo/europython$
   python3 compare.py
11 True
```

## Comparing files and directories

```
1 dir1 = "./dir1"  
2 dir2 = "./dir2"  
3  
4 cmp_dirs = filecmp.dircmp(dir1, dir2)  
5 cmp_dirs.report()
```

## Comparing files and directories

```
1 dir1 = "./dir1"
2 dir2 = "./dir2"
3
4 cmp_dirs = filecmp.dircmp(dir1, dir2)
5 cmp_dirs.report()
6
7 mia@mias-
  ntb:~/Documents/repos/osobní/python/demo/europython$
  python3 compare.py
8 diff ./dir1 ./dir2
9 Only in ./dir1 : ['file1.py']
10 Only in ./dir2 : ['file2.py']
```

## Comparing files and directories

- the `filecmp` module defines functions to compare files



## Comparing files and directories

- the `filecmp` module defines functions to compare files
- the `dircmp` class constructs a new directory comparison object `co` to compare two directories and provides multiple functions to define what to compare and how to show the results

## Comparing files and directories

- the `filecmp` module defines functions to compare files
- the `dircmp` class constructs a new directory comparison object `co` to compare two directories and provides multiple functions to define what to compare and how to show the results
- the `filecmp()` module is useful for cases where we have different versions of the same project

Context manager

## Context manager

```
1 from contextlib import contextmanager
2
3
4 @contextmanager
5 def managed_file(name):
6     try:
7         f = open(name, "w")
8         print("Opened the file: ", name)
9         yield f
10    finally:
11        f.close()
12        print("Closed the file: ", name)
13
14
15 with managed_file("hello.txt") as f:
16     f.write("Hello world!")
17     print("Wrote to file")
```

## Context manager

```
1 mia@mias-  
  ntb:~/Documents/repos/osobní/python/demo/europython$  
  python3 context_manager.py  
2 Opened the file:  hello.txt  
3 Wrote to file  
4 Closed the file:  hello.txt
```

## Context manager

- defines a factory function for 'with' statement contexts

## Context manager

- defines a factory function for 'with' statement contexts
- provides a clean, easy-to-read way to manage resources that need setup and teardown phases

## Context manager

- defines a factory function for 'with' statement contexts
- provides a clean, easy-to-read way to manage resources that need setup and teardown phases
- you can nest multiple context managers with blocks to use them at once or use in a single with statement by separating them with commas



## Context manager

- for opening and closing files, the built-in function `open()` handles it, but other common use-cases are:

## Context manager

- for opening and closing files, the built-in function `open()` handles it, but other common use-cases are:
  - acquiring and releasing a lock

## Context manager

- for opening and closing files, the built-in function `open()` handles it, but other common use-cases are:
  - acquiring and releasing a lock
  - working with network connections

## Context manager

- for opening and closing files, the built-in function `open()` handles it, but other common use-cases are:
  - acquiring and releasing a lock
  - working with network connections
  - temporary files

## Context manager

- for opening and closing files, the built-in function `open()` handles it, but other common use-cases are:
  - acquiring and releasing a lock
  - working with network connections
  - temporary files
  - changing and restoring global settings

## Context manager

```
1 import aiofiles
2 from contextlib import asynccontextmanager
3 import asyncio
4
5
6 @asynccontextmanager
7 async def managed_file(name):
8     try:
9         f = await aiofiles.open(name, "w")
10        print("Opened the file: ", name)
11        yield f
12    finally:
13        await f.close()
14        print("Closed the file: ", name)
15
```

## Context manager

```
1 async def main():
2     async with managed_file("hello.txt") as f:
3         await f.write("Hello world!")
4         print("Wrote to file")
5
6
7 asyncio.run(main())
```

## Conclusion



## Conclusion

- the standard library is packed with very powerful tools

## Conclusion

- the standard library is packed with very powerful tools
- always verify if there's an existing tool for your task and if it suits your needs before attempting to create something from scratch

Thank you for your attention!

- slides: [github.com/clytaemnestra/talks](https://github.com/clytaemnestra/talks)
- contact me:
  - LinkedIn: [linkedin.com/in/mia-bajic](https://www.linkedin.com/in/mia-bajic)
  - email: [miabajic.miabajic@gmail.com](mailto:miabajic.miabajic@gmail.com)

## Q&A

- slides: [github.com/clytaemnestra/talks](https://github.com/clytaemnestra/talks)
- contact me:
  - LinkedIn: [linkedin.com/in/mia-bajic](https://www.linkedin.com/in/mia-bajic)
  - email: [miabajic.miabajic@gmail.com](mailto:miabajic.miabajic@gmail.com)