The Standard Library Tour

Mia Bajic

link-to-notes

linkedin.com/in/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?

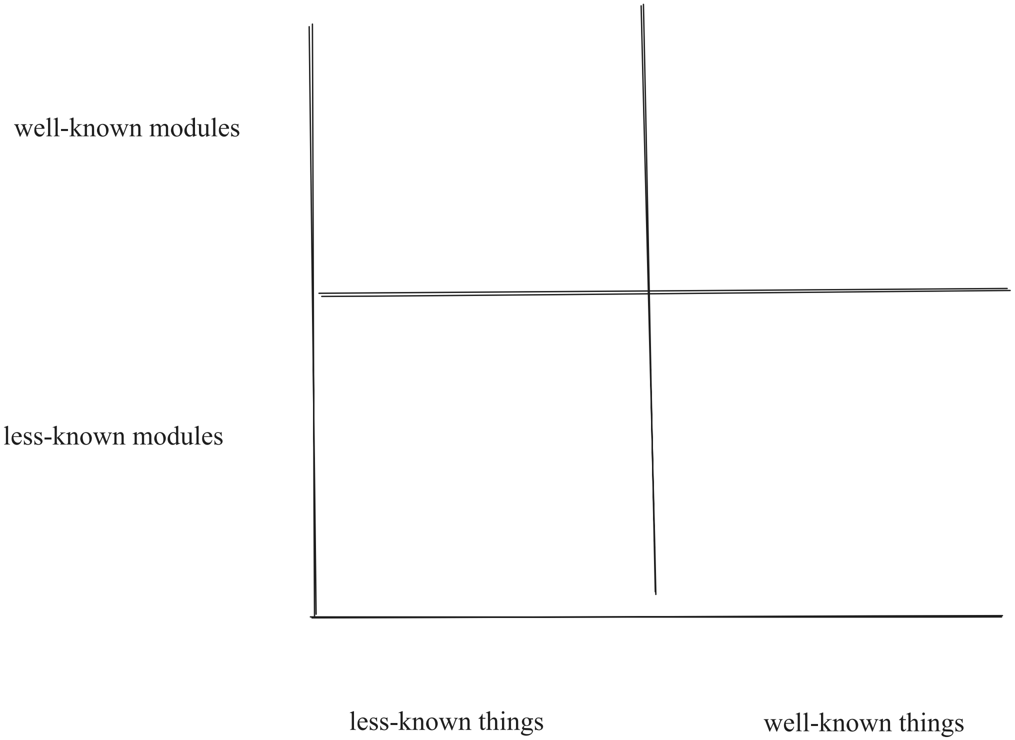
What is this talk about?

- 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?



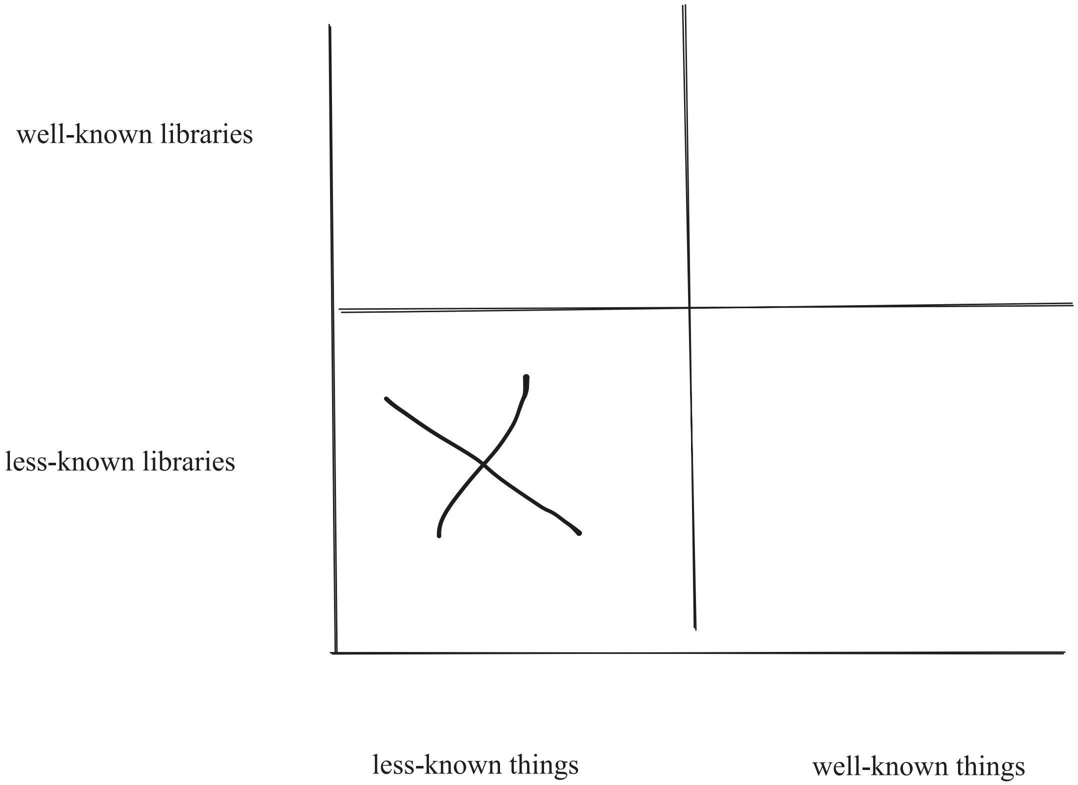
well-known modules

less-known modules

less-known things

well-known things

What is this talk about?



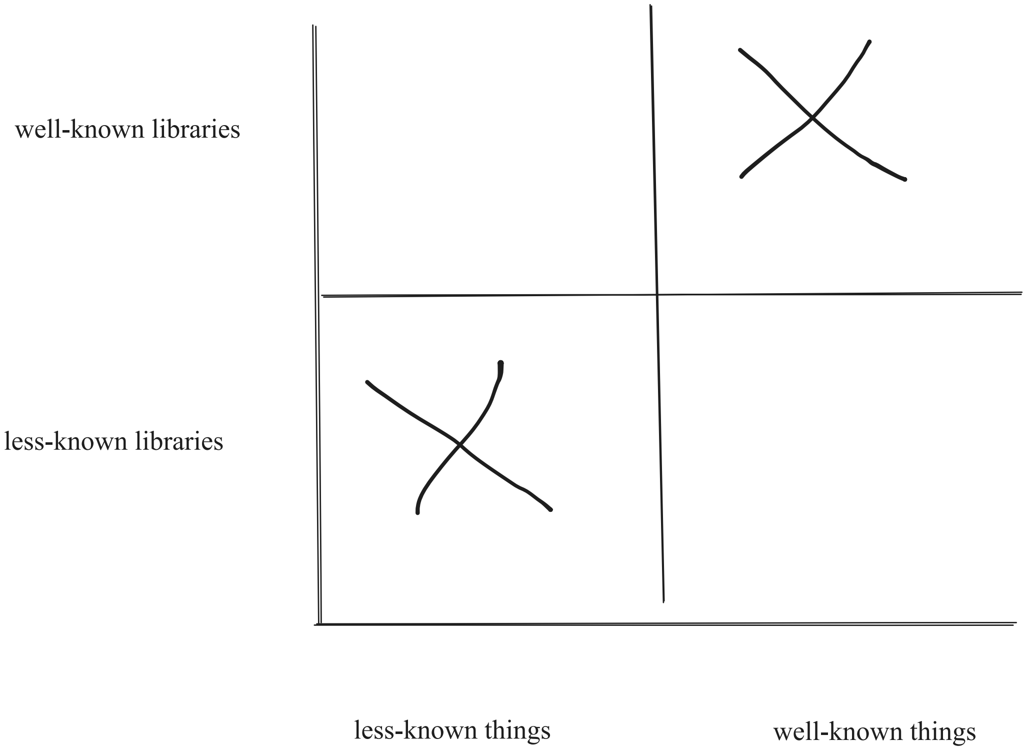
well-known libraries

less-known libraries

less-known things

well-known things

What is this talk about?



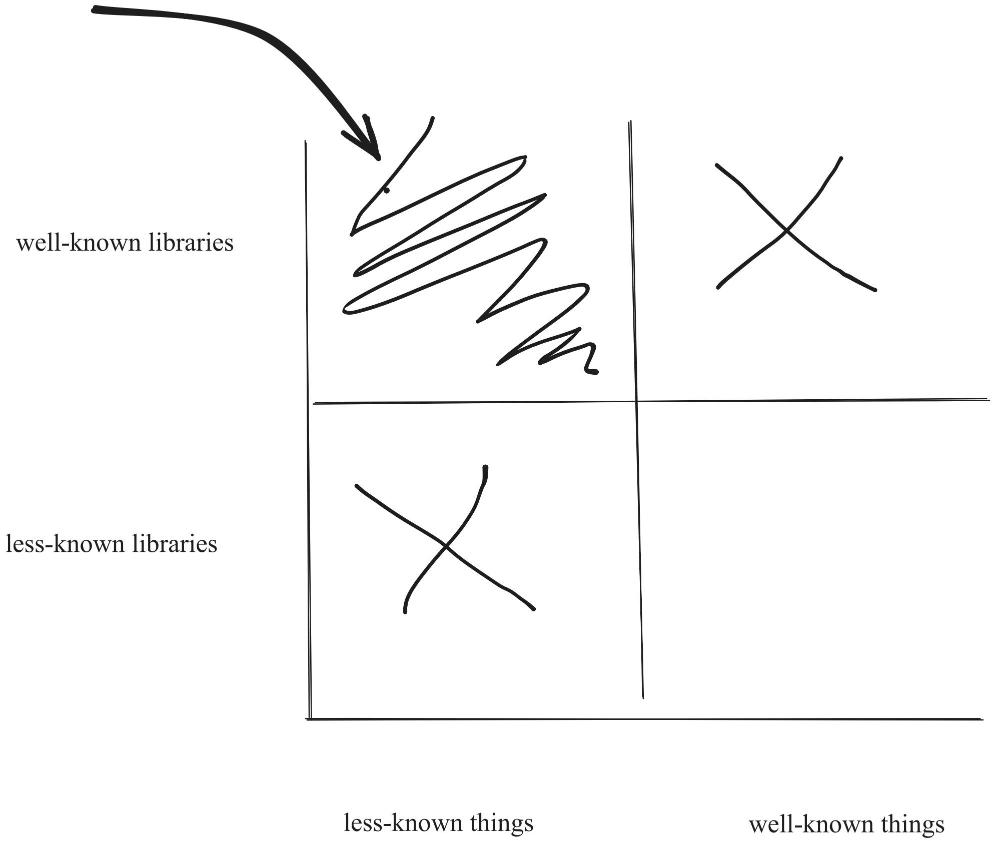
well-known libraries

less-known libraries

less-known things

well-known things

Well-known libraries that do less-known things



well-known libraries

less-known libraries

less-known things

well-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

```python
def add(x, y):
    if isinstance(x, int) and isinstance(y, int):
        return x + y
    elif isinstance(x, str) and isinstance(y, str):
        return x + " " + y
    else:
        raise ValueError("Unsupported data type")
```

Functools

```python
def add(x, y):
    if isinstance(x, int) and isinstance(y, int):
        return x + y
    elif isinstance(x, str) and isinstance(y, str):
        return x + " " + y
    else:
        raise ValueError("Unsupported data type")

print(add(1, 2))
print(add("Hello", "World!"))
print(add(1, "World!"))
```

Functools

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

Functools

```python
from functools import singledispatch


@singledispatch
def add(x, y):
    raise ValueError("Unsupported data type")


@add.register
def _(x: int, y: int):
    return x + y


@add.register
def _(x: str, y: str):
    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

```python
def add_to_two(x, y=2):
    return x + y

print(add_to_two(3))
```

Functools

```python
def add_to_three(x, y=3):
    return x + y

print(add_to_three(2))
```

Functools

```python
import functools


def add(x, y):
    return x + y

add_to_two = functools.partial(add, y=2)
add_to_three = functools.partial(add, y=3)

print(add_to_two(3))
print(add_to_three(3))
```

Functools

```python
import functools


def add(x, y):
    return x + y

add_to_two = functools.partial(add, y=2)
add_to_three = functools.partial(add, y=3)

print(add_to_two(3))
print(add_to_three(3))

5
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

```
import functools

def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Functools

```python
import functools

@functools.lru_cache(maxsize=None)
def fibonacci(n):
    if n < 2:
        return n
    return fibonacci(n-1) + fibonacci(n-2)
```

Functools

```python
import functools

@functools.cache
def fibonacci(n):
    if n < 2:
        return n
    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 calls a lru_cache_wrapper, which wraps the function

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
- if the key is missing, the wrapper calls the user function with passed arguments, updates the cache miss info and returns the result

Functools

- 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
- if the key is missing, the wrapper calls the user function with passed arguments, updates the cache miss info and returns the result
- if the cache is full, it 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

```
mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 -mtimeit -s 'import fibonacci_without_cache as f' 'f.fibonacci(n=20)'
200 loops, best of 5: 1.03 msec per loop

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 -mtimeit -s 'import fibonacci_with_lru_cache as f' 'f.fibonacci(n=20)'
5000000 loops, best of 5: 97.1 nsec per loop

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 -mtimeit -s 'import fibonacci_with_cache as f' 'f.fibonacci(n=20)'
2000000 loops, best of 5: 97.8 nsec per loop
```

Itertools

Itertools

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

Itertools

```
list1 = [1, 2, 3]
list2 = [4, 5, 6]

product = []

for i in list1:
    for j in list2:
        product.append((i, j))
print(product)

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 product.py
[(1, 4), (1, 5), (1, 6), (2, 4), (2, 5), (2, 6), (3, 4), (3, 5), (3, 6)]
```

Itertools

```
import itertools

list1 = [1, 2, 3]
list2 = [4, 5, 6]

print(list(itertools.product(list1, list2)))

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 product.py
[(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

```
list1 = [1, 2, 3, 4]
list2 = [4, 5, 6]

print(list(filter(lambda i: i in list1, list2)))

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 filter.py
[4]
```

Itertools

```
import itertools


list1 = [1, 2, 3, 4]
list2 = [4, 5, 6]

print(list(itertools.filterfalse(lambda i: i in list1, list2)))

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 filter.py
[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

```python
numbers = [1, 2, 3]
letters = ['a', 'b', 'c', 'd', 'e']

print(list(zip(numbers, letters)))
```

Itertools

```
numbers = [1, 2, 3]
letters = ['a', 'b', 'c', 'd', 'e']

print(list(zip(numbers, letters)))

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 zip.py
[(1, 'a'), (2, 'b'), (3, 'c')]
```

Itertools

```
import itertools

numbers = [1, 2, 3]
letters = ['a', 'b', 'c', 'd', 'e']

print(list(itertools.zip_longest(numbers, letters)))

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 zip_longest.py
[(1, 'a'), (2, 'b'), (3, 'c'), (None, 'd'), (None, 'e')]
```

Collections

Collections

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

Collections

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

if "c" in dict1:
    print(dict1["c"])
elif "c" in dict2:
    print(dict2["c"])
else:
    print("Not found")

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 chainmap.py
3
```

Collections

```
from collections import ChainMap

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

chain_dict = ChainMap(dict1, dict2)
print(chain_dict["c"])

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 chainmap.py
3
```

Collections

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

new_dict = {}
new_dict.update(dict1)
new_dict.update(dict2)

print(new_dict)

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 chainmap.py
{'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

```python
dict1 = {"a": 1, "b": 2, "c": 3}

print(dict1["d"])
```

Collections

```
dict1 = {"a": 1, "b": 2, "c": 3}

print(dict1["d"])

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 default_dict.py
Traceback (most recent call last):
  File "/home/mia/Documents/repos/osobní/python/demo/europython/default_dict.py", line 3, in <module>
    print(dict1["d"])
KeyError: 'd'
```

Collections

```python
from collections import defaultdict

dict1 = defaultdict(lambda: None, {"a": 1, "b": 2, "c": 3})

print(dict1["d"])
```

Collections

```
from collections import defaultdict

dict1 = defaultdict(lambda: None, {"a": 1, "b": 2, "c": 3})

print(dict1["d"])

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 default_dict.py
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

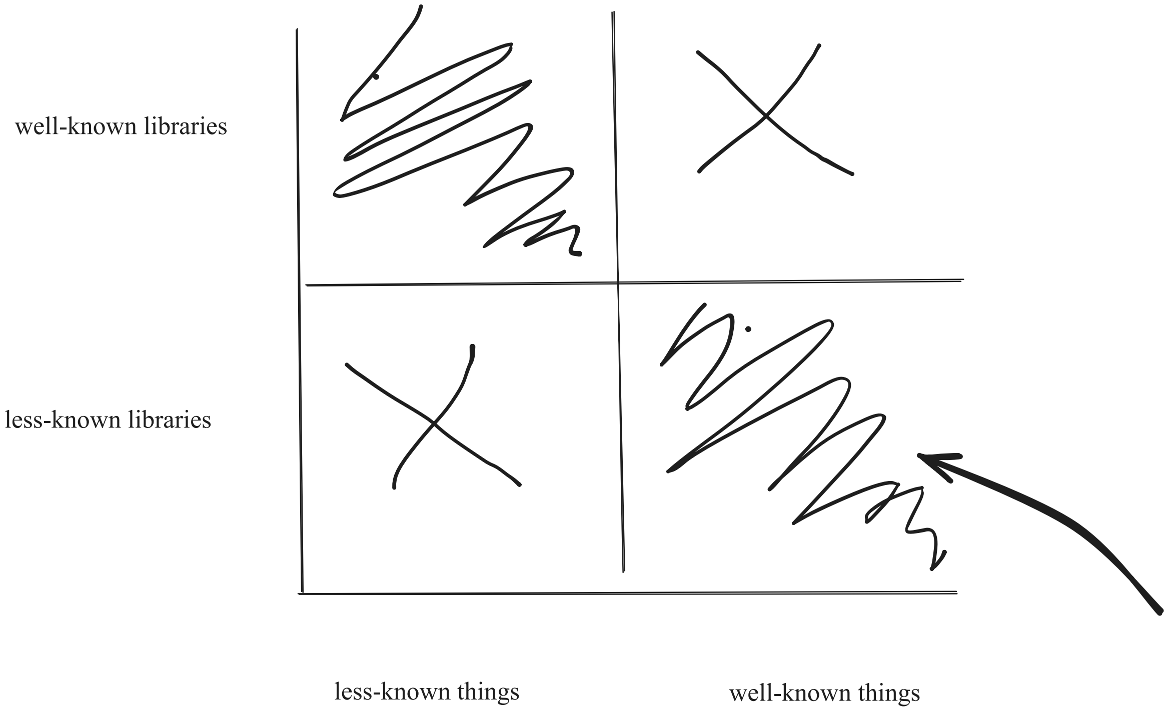
Collections

```
dict2 = defaultdict(lambda: None, defaultdict(lambda: None, {"a": 1, "b": 2, "c": 3}))

print(dict2["d"])

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 default_dict.py
None
```

Less-known modules which do well-known things



well-known libraries

less-known libraries

less-known things

well-known things

Testing

Testing

```python
def add(x, y):
    return x + y
```

Testing

```python
def add(x, y):
    """
    Adds the two input numbers.

    >>> add(2, 3)
    5
    >>> add(-1, 1)
    0
    >>> add(-10, -5)
    -15
    """
    return x + y

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Testing

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

```

Testing

```python
def add(x, y):
    """
    Adds the two input numbers.

    >>> add(2, 3)
    5
    >>> add(-1, 1)
    0
    >>> add(-10, -5)
    15  # this line changed
    """
    return x + y

if __name__ == "__main__":
    import doctest
    doctest.testmod()
```

Example

```
mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 testmod.py
**********************************************************************
File "/home/mia/Documents/repos/osobní/python/demo/europython/testmod.py", line 9, in __main__.add
Failed example:
    add(-10, -5)
Expected:
    15
Got:
    -15
**********************************************************************
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

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

Comparing sequences

```python
import difflib
from pprint import pprint

string1 = "Hello world!"
string2 = "Hello World!"

d = difflib.Differ()
result = list(d.compare(string1, string2))

pprint(result)
```

Comparing sequences

```
mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 differ.py
['  H',
 '  e',
 '  l',
 '  l',
 '  o',
 '   ',
 '- w',
 '+ W',
 '  o',
 '  r',
 '  l',
 '  d',
 '  !']
```

Comparing sequences

```python
import difflib
from pprint import pprint

string1 = "Hello world!"
string2 = "Hello World!"

s = difflib.SequenceMatcher(None, string1, string2)
print(s.ratio())
```

Comparing sequences

```
import difflib
from pprint import pprint

string1 = "Hello world!"
string2 = "Hello World!"

s = difflib.SequenceMatcher(None, string1, string2)
print(s.ratio())

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 differ.py
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

```python
import filecmp

file1 = "./dir1/file1.py"
file2 = "./dir2/file2.py"

cmp = filecmp.cmp(file1, file2)

print(cmp)
```

Comparing files and directories

```
import filecmp

file1 = "./dir1/file1.py"
file2 = "./dir2/file2.py"

cmp = filecmp.cmp(file1, file2)

print(cmp)

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 compare.py
True
```

Comparing files and directories

```
dir1 = "./dir1"
dir2 = "./dir2"

cmp_dirs = filecmp.dircmp(dir1, dir2)
cmp_dirs.report()
```

Comparing files and directories

```
dir1 = "./dir1"
dir2 = "./dir2"

cmp_dirs = filecmp.dircmp(dir1, dir2)
cmp_dirs.report()

mia@mias-ntb:~/Documents/repos/osobní/python/demo/europython$ python3 compare.py
diff ./dir1 ./dir2
Only in ./dir1 : ['file1.py']
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 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 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

```python
from contextlib import contextmanager


@contextmanager
def managed_file(name):
    try:
        f = open(name, "w")
        print("Opened the file: ", name)
        yield f
    finally:
        f.close()
        print("Closed the file: ", name)


with managed_file("hello.txt") as f:
    f.write("Hello world!")
    print("Wrote to file")
```

Context manager

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

```python
import aiofiles
from contextlib import asynccontextmanager
import asyncio


@asynccontextmanager
async def managed_file(name):
    try:
        f = await aiofiles.open(name, "w")
        print("Opened the file: ", name)
        yield f
    finally:
        await f.close()
        print("Closed the file: ", name)

```

Context manager

```
async def main():
    async with managed_file("hello.txt") as f:
        await f.write("Hello world!")
        print("Wrote to file")


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: xxx
- contact me:
  - LinkedIn: linkedin.com/in/mia-bajic
  - email: miabajic.miabajic@gmail.com

Q&A

- slides: xxx
- contact me:
  - LinkedIn: linkedin.com/in/mia-bajic
  - email: miabajic.miabajic@gmail.com