

Oral Exam Synopsis

By Andreas Møller Laursen

Data Structures & Functions

In my presentation for this topic, I will firstly **show** the **four different data structures** built-in in Python. These consist of **lists**, **tuples**, **sets** and **dictionaries**. After having shown the data structures and how they operate, I will explain the **main differences** between the data structures and when to use each (A list of the different data structure is attached at the end of this file). After having shown and talked about data structures I wanna show how to make a **function**, and then how to **call a function within a function**, so that I can showcase a case of **inner classes** within Python, and how we can make **versatile code**. After going quickly through this, I wanna also show how we can implement the built in function **.sorted()** in our code, and how we actually use this tool. I will also show how we can **finetune** this function to best suit our needs. Lastly I wanna show **list comprehensions** and how they work, both with conditions and without conditions

```
(i for i in range(10))
```

← without condition

```
(x ** 2 for x in range(10) if x % 2 == 0)
```

← with condition

Pythonic OOP

In my presentation I will show the creation of a **small program via OOP** (this will probably be the bank exercise we did during session 6). Here I will show how to **create classes & their class variables and functions**. After this I will show how to **instantiate a class**. When all of this is working, I will implement inheritance in my small program, probably by extending an already existing class, then showing off how inheritance works, and how it can help us write improved code.

When this is done I will explain **why OOP** is important to us, and **when to use** it instead of procedural programming. I will also give a **quick comparison** between Pythonic OOP and Java OOP

After having done this, I will show off some **encapsulation**, by converting our public attributes to **private attributes**, then explain what it means to be private in Python.

If there's time, I will show off how to **implement the @property** decorator in our code.

Generators, Decorators & Context Managers

During this section, I will firstly talk about generators and how we can make our own **classes iterable**. I will talk about this example

```
class Compute:
    def __iter__(self):
        self.last = 0
        return self

    def __next__(self):
        rv = self.last
        self.last += 1
        if self.last > 10:
            raise StopIteration()
        sleep(.5)
        return rv


for i in Compute():
    print(i)
```

and how this makes it possible for us to use a for loop without breaking the code during run-time, and the general inner workings of the **iterator**. I will also talk about this snippet of code

```
def compute():
    for i in range(10):
        yield i
```

and how this makes it possible for us to use **next()**

After this I will show off a quick example of a **generator expression**, probably with this example



```
string = 'geek'
li = list(string[i] for i in range(len(string)-1, -1, -1))
print(li)
```

Output:

```
['k', 'e', 'e', 'g']
```

After all this I will talk about **decorators** in Python, these consist of **@decorator** and **@property**

First I will show off how Python makes it possible to have **first class functions** and **inner functions**, by showcasing an example where I pass a lambda to print out a returned value from a function, by passing a string and another function.

Then I will show decorators in action, and how it can help us manipulate data within a function before it is returned, making it possible for us to outsource tasks to other functions within the same class, making the code more readable, by using the **@decorator** decorator, then showcasing a case where I implement the **@property** decorator

Lastly I will showcase the use of **Context Managers**, but since we haven't been over this during classes, this will be built upon when we get closer to the exam and I've actually read up on the subject.

Python Modules & the Python Development Environment

Here I will show off how we can implement code other people have written for us, by **importing modules** and using these modules in our code. I will probably go through the **web scraper** we did during session session 5, exercise 6. Here I use the importing of both the **requests** module, but also **BeautifulSoup**. I will explain the **advantages** of using modules within python, and also where you can find new modules to import (<https://pypi.org/>).

Within the scope of this project, I will also show off how you can **dockerise** your project, by having a **DockerFile** and a **requirements.txt** file within your project. I will showcase how you can **create** these, **use** them, **upload** them to Github, then show how people on the other end (those who extract your project from Github) can use these to quickly have the exact same **project environment** as you. Thus we escape the problems of working in different environments. This part will honestly take a long time, so I doubt I'll have time for further explanations after this.

List of different Data Structures in Python

Lists

Declaration of a list

```
mylist = ["apple", "banana", "cherry"]
```

Attributes of a list

- Ordered list, all elements will remain at the same spots
- Changeable, items within the list can be added, removed or changed
- Duplicates, multiple of the same value in a list is allowed
- Any data type, everything within the list can be of any data type

Tuples

Declaration of a tuple

```
mytuple = ("apple", "banana", "cherry")
```

Attributes of a tuple

- Ordered list, all elements will remain at the same spots
- Immutable, items within the tuple cant be added, removed or changed
- Duplicates, multiple of the same value in a tuple is allowed
- Any data type, everything within the tuple can be of any data type

Sets

Declaration of a tuple

```
myset = {"apple", "banana", "cherry"}
```

Attributes of a tuple

- Unordered list, items will be presented randomly, thus reference via index is not a valid option
- Immutable, items within the tuple cant be added, removed or changed
- No dupes, multiple of the same value is not allowed within a dict
- Any data type, everything within the tuple can be of any data type

Dictionary

Declaration of a tuple

```
thisdict = {  
    "brand": "Ford",  
    "model": "Mustang",  
    "year": 1964  
}
```

Attributes of a tuple

- Ordered list, all elements will remain at the same spots
- Mutable, items within the dict can be added, removed or changed
- No dupes, multiple of the same value is not allowed within a dict
- Any data type, everything within the tuple can be of any data type