# Data Structures & functions

* List
  + Data structure that is mutable an ordered sequence of elements
  + Example info <https://www.digitalocean.com/community/tutorials/understanding-lists-in-python-3>
    - Will show examples of different list functions
* Tuple
  + A tuple is a sequence of immutable Python objects. Tuples are sequences, just like lists. The differences between tuples and lists are, the tuples cannot be changed unlike lists and tuples use parentheses, whereas lists use square brackets.
  + Example info <https://www.tutorialspoint.com/python/python_tuples.htm>
* Set
  + A set is a collection which is unordered and unindexed. In Python sets are written with curly brackets.
  + Example info <https://www.w3schools.com/python/python_sets.asp>
* Dictionaries
  + **Dictionary**in Python is an unordered collection of data values, used to store data values like a map, which unlike other Data Types that hold only single value as an element, Dictionary holds **key:value** pair. Key value is provided in the dictionary to make it more optimized. Each key-value pair in a Dictionary is separated by a colon **:**, whereas each key is separated by a ‘comma’.
  + <https://www.geeksforgeeks.org/python-dictionary/>
* Slicing
  + **The slice() function returns a slice object that can use used to slice strings, lists, tuple etc.** The slice object is used to slice a given sequence ([string](https://www.programiz.com/python-programming/string), [bytes](https://www.programiz.com/python-programming/methods/built-in/bytes), [tuple](https://www.programiz.com/python-programming/tuple), [list](https://www.programiz.com/python-programming/list)or [range](https://www.programiz.com/python-programming/methods/built-in/range)) or any object which supports sequence protocol (implements \_\_getitem\_\_() and \_\_len\_\_() method).
  + <https://www.programiz.com/python-programming/methods/built-in/slice>
* Sorting
  + The sort function can be used to sort a list in ascending, descending or user defined order.
  + <https://www.geeksforgeeks.org/python-list-sort/>

# IO & Context Managers

* Filetypes
  + Find ud af de 4-5 mest brugte fil typer og hvis hvordan de bruges i python
* Context Managers
  + **Managing Resources :** In any programming language, the usage of resources like file operations or database connections is very common. But these resources are limited in supply. Therefore, the main problem lies in making sure to release these resources after usage. If they are not released then it will lead to resource leakage and may cause the system to either slow down or crash. It would be very helpful if user have a mechanism for the automatic setup and teardown of resources.In Python, it can be achieved by the usage of context managers which facilitate the proper handling of resources. The most common way of performing file operations is by using the with keyword as shown below:
  + <https://www.geeksforgeeks.org/context-manager-in-python/>

My presentation for IO & Context Managers will incorperate theoretical information about filtypes and

their use in Python, as well a code demonstration of how you can use Python to read from and write to the

files furthermore, I will inform about Context Managers, why they are good to use and how they work.

# Pythonic OOP

* Classes and Objects
  + Python is an object oriented programming language. Unlike procedure oriented programming, where the main emphasis is on functions, object oriented programming stress on objects.

Object is simply a collection of data (variables) and methods (functions) that act on those data. And, class is a blueprint for the object.

We can think of class as a sketch (prototype) of a house. It contains all the details about the floors, doors, windows etc. Based on these descriptions we build the house. House is the object.

As, many houses can be made from a description, we can create many objects from a class. An object is also called an instance of a class and the process of creating this object is called **instantiation**.

* + <https://www.programiz.com/python-programming/class>
* Inheritance and Encapsulation
  + Inheritance is a powerful feature in object oriented programming.

It refers to defining a new [class](https://www.programiz.com/python-programming/class) with little or no modification to an existing class. The new class is called **derived (or child) class** and the one from which it inherits is called the **base (or parent) class**.

* + - <https://www.programiz.com/python-programming/inheritance>
  + Encapsulation is one of the fundamental concepts in object-oriented programming (OOP). It describes the idea of wrapping data and the methods that work on data within one unit. This puts restrictions on accessing variables and methods directly and can prevent the accidental modification of data. To prevent accidental change, an object’s variable can only be changed by an object’s method. Those type of variables are known as **private varibale**.
    - <https://www.geeksforgeeks.org/encapsulation-in-python/>
* \_\_repr\_\_ & \_\_str\_\_
  + According to the official Python documentation, \_\_repr\_\_ is a built-in function used to compute the "official" string reputation of an object, while \_\_str\_\_ is a built-in function that computes the "informal" string representations of an object. So both \_\_repr\_\_ and \_\_str\_\_ are used to represent objects, but in different ways. The best way to understand the difference between these two functions is to see them in action:
    - <https://www.pythonforbeginners.com/basics/__str__-vs-__repr>

My presentation for Pythonic OOP will incorperate theoretical information about Classes and Objects,

Inheritence and Encapsulation as well as a code demonstration of how you can use these concepts in your

code, furthermore I will inform about \_\_repr\_\_ and \_\_str\_\_ methods.

# \_\_Protocols\_\_

* what it means that Python is a protocol based language spørg anders??
  + In these languages, a protocol is a common means for discrete [objects](https://en.wikipedia.org/wiki/Object_(computer_science)) to communicate with each other. These are definitions of [methods](https://en.wikipedia.org/wiki/Method_(computer_science)) and values which the objects agree upon, in order to co-operate, as part of an [API](https://en.wikipedia.org/wiki/Application_programming_interface).

The protocol/interface is a description of:

* + - The messages that are understood by the object.
    - The arguments that these messages may be supplied with.
    - The types of results that these messages return.
    - The [invariants](https://en.wikipedia.org/wiki/Invariant_(computer_science)) that are preserved despite modifications to the state of an object.
    - The exceptional situations that will be required to be handled by clients to the object.
    - (For the communications-style usage only:) The call sequence and decision points of the methods, such as would be representing in UML interaction diagrams: [Communication diagram](https://en.wikipedia.org/wiki/Communication_diagram), [Sequence diagram](https://en.wikipedia.org/wiki/Sequence_diagram), [Interaction overview diagram](https://en.wikipedia.org/wiki/Interaction_overview_diagram)/[Activity diagram](https://en.wikipedia.org/wiki/Activity_diagram), [Timing diagram](https://en.wikipedia.org/wiki/Timing_diagram_(Unified_Modeling_Language)).
  + <https://en.wikipedia.org/wiki/Protocol_(object-oriented_programming)>
* Context Managers
  + **Managing Resources :** In any programming language, the usage of resources like file operations or database connections is very common. But these resources are limited in supply. Therefore, the main problem lies in making sure to release these resources after usage. If they are not released then it will lead to resource leakage and may cause the system to either slow down or crash. It would be very helpful if user have a mechanism for the automatic setup and teardown of resources.In Python, it can be achieved by the usage of context managers which facilitate the proper handling of resources. The most common way of performing file operations is by using the with keyword as shown below:
  + <https://www.geeksforgeeks.org/context-manager-in-python/>
* Iterator
  + An iterator is an object that contains a countable number of values.
  + An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
  + Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods \_\_iter\_\_() and \_\_next\_\_().
    - <https://www.w3schools.com/python/python_iterators.asp>
* Top-Level functions
  + Snak med anders=?

My presentation for \_\_ Protocols \_\_ I will explain what it means that Python is a protocol based language,

and will give theoretical information about Context Managers and Iterator, as well as a code demonstration

for this, furthermore i will inform about Top-level functions & their \_\_functions\_\_ implementations.

# Generators

* Generators
  + There is a lot of overhead in building an [iterator in Python](https://www.programiz.com/python-programming/iterator); we have to implement a class with \_\_iter\_\_() and \_\_next\_\_() method, keep track of internal states, raise StopIteration when there was no values to be returned etc.

This is both lengthy and counter intuitive. Generator comes into rescue in such situations.

Python generators are a simple way of creating iterators. All the overhead we mentioned above are automatically handled by generators in Python.

Simply speaking, a generator is a function that returns an object (iterator) which we can iterate over (one value at a time).

* + <https://www.programiz.com/python-programming/generator>
* Iterator
  + An iterator is an object that contains a countable number of values.
  + An iterator is an object that can be iterated upon, meaning that you can traverse through all the values.
  + Technically, in Python, an iterator is an object which implements the iterator protocol, which consist of the methods \_\_iter\_\_() and \_\_next\_\_().
    - <https://www.w3schools.com/python/python_iterators.asp>
* Contextlib

|  |  |
| --- | --- |
| * Purpose: | Utilities for creating and working with context managers. |

the contextlib module contains utilities for working with context managers and the with statement.

* + <https://pymotw.com/3/contextlib/>
* @contextmanager
  + ?? snak med anders
  + <https://jeffknupp.com/blog/2016/03/07/python-with-context-managers/>
* Contextdecorator (might be decorator and not context decorator)
  + Example?? <https://coderwall.com/p/0lk6jg/python-decorators-vs-context-managers-have-your-cake-and-eat-it>
  + Python decorators are a powerful concept that allow you to "wrap" a function with another function.

The idea of a decorator is to abstract away something that you want a function or class to do, besides its normal responsibility. This can be helpful for many reasons such as code reuse, and sticking to [curlys law](https://blog.codinghorror.com/curlys-law-do-one-thing/).

By learning how to write your own decorators, you can significantly improve readability of your own code. They can change how the function behaves, without needing to actually change the code (such as adding logging lines). They are a fairly common tool in python, familiar to those who use frameworks such as flask or click, but many only know how to use them, not how to write their own.

* + - <https://timber.io/blog/decorators-in-python/>

My presentation for Generators will incorperate theoretical information about Generators & the Iterator

class, as well as show code examples on how these can be used, furthermore I will inform about Contextlib

with @contextmanager and ContextDecorator.

# Decorators

* 1. *Example??* [*https://coderwall.com/p/0lk6jg/python-decorators-vs-context-managers-have-your-cake-and-eat-it*](https://coderwall.com/p/0lk6jg/python-decorators-vs-context-managers-have-your-cake-and-eat-it)
  2. *Python decorators are a powerful concept that allow you to "wrap" a function with another function.*

*The idea of a decorator is to abstract away something that you want a function or class to do, besides its normal responsibility. This can be helpful for many reasons such as code reuse, and sticking to [curlys law](https://blog.codinghorror.com/curlys-law-do-one-thing/).*

*By learning how to write your own decorators, you can significantly improve readability of your own code. They can change how the function behaves, without needing to actually change the code (such as adding logging lines). They are a fairly common tool in python, familiar to those who use frameworks such as flask or click, but many only know how to use them, not how to write their own.*

* + 1. [*https://timber.io/blog/decorators-in-python/*](https://timber.io/blog/decorators-in-python/)
* Snak med anders

My presentation for Decorators will incorperate theoretical information about inner functions, simple decorators and the decorator pattern, as well as show code examples on how these can be used in Python, furthermore, I will inform about return values from decorators