# FoLT Tutorial 1 Summary

## *PART1: Basic Python*

- Things in python to keep in mind:
- *Lists*:
  - The `append` method adds a new element at the end of the original list. The original list is changed and no new object is created. So there is no return value.
  - The `+` operator concatenates two lists together and returns a new object. The original object remains unchanged.
- *Formatting*:

```
        print("Hello " + y + "!")
        print("Hello {}!".format(y)) # the value of `y` will be inserted into the
placeholders, denoted by `{}`
        print(f"Hello {y}!")
```

- *Sets*:
  - Sets are unordered, unchangeable, and not indexed. Items in a list have a defined order. The items of a set cannot be modified once the set has been created, but we can remove items and add new items. Sets do not allow duplicates.
  - Example:

```
                my_set = {1,4,8,9,0}
                print(type(my_set))
"""Output: <class 'set'>"""
                my_set.add(3)
                my_set.remove(1)
                print(my_set)
"""Output: {0, 3, 4, 8, 9}"""
```

- *Dicts*:
  - A dictionary is used to store key-value pairs. Dictionaries in Python do not allow duplicates and can not be accessed or modified through index.
  - Example:

```
                student_dict = {
                "name": "Mustermann",
                "first name": "Max",
                "ID": 1234
                }
                print(student_dict)
#Output: {'name': 'Mustermann', 'first name': 'Max', 'ID': 1234}
                student_dict["age"] = 23
                student_dict["first name"] = "Mia"
                print(student_dict)
#Output: {'name': 'Mustermann', 'first name': 'Mia', 'ID': 1234, 'age': 23}
                student_dict.update({"first name": "Mia"})
                print(student_dict)
#Output: {'name': 'Mustermann', 'first name': 'Mia', 'ID': 1234, 'age': 23}
                student_dict.pop("ID") #`pop()` removes the element with the specified
```

```
key name from the dictionary. Use this to remove the "ID" from the dictionary.
#Output: {'name': 'Mustermann', 'first name': 'Mia', 'age': 23}
```

- Looping through a Dict:

```
for a in student_dict:
    print(a)
print("---")
for b in student_dict:
    print(student_dict[b])
print("---")
for c in student_dict.values():
    print(c)
print("---")
for d in student_dict.keys():
    print(d)
print("---")
for e1, e2 in student_dict.items():
print(e1,e2)

#Output:
"""
name
first name
age
---
Mustermann
Mia
23
---
Mustermann
Mia
23
---
name
first name
age
---
name Mustermann
first name Mia
age 23
"""
```

## *PART 2: Basic NLTK*

- *Definition:* The Natural Language Toolkit is a platform for working with NLP in Python. It provides libraries with datasets, and functionalities like tokenization, stemming, and parsing.
    - NLTK is a huge library, so we are gonna play with some of its functions and comment their output and their utility

```
import nltk #import library
nltk.download() #download the files
from nltk.book import * #NLTK book dataset
nltk.book.texts() #the text inside the books
text1.count("book") #count the occurnce of book in text 1
```

```
        text1.concordance("book") #Gives back the context where book appears
        text1.similar("book") #Gives back words that are similar in context and usage to
"book"
```

- Some functions to tokenize and count tokens, make a vocabulary...

```
        len(text1) #Output the number of tokens of `text1`
        len(set(text1)) #Now, output the size of the vocabulary of `text1`

        #Getting the top 10 most common words using the Counter libraries
        from collections import Counter
        c = Counter(text1)
        print(c.most_common(10))

        #Using dicts to do the same thing without Counter
        occurancy_dict = {}
        for word in text1:
            if word in occurancy_dict:
                occurancy_dict[word] += 1
            else:
                occurancy_dict[word] = 1
        sorted(occurancy_dict.items(), key=lambda x: x[1], reverse=True)[:10] #Use key
as the number of repetitions x[1] and reverse=True so the sorting would be from more
reps to less reps
```

- NLTK functions on sentences:

```
        nltk.book.sents() #Sentences can be stored as lists of strings.
        text4.collocations() #A collocation is a sequence of words that occur together
unusually often.
```

## PART 3: Functions and fun Stuff:

- Functions are really easy to make in python, so I'm only gonna give examples of lambda functions:
  - **Lambda functions** or anonymous functions are a shorthand to define functions with only one statement.
- Note that lambda functions do not use a function header or a return statement!
  - In this example, a lambda expression is used to concatenate variables a, b, and c and return the result:

```
        concatenate_three = lambda a, b, c : a + b + c
        print(concatenate_three("Hi ", "Mr. ", "Brown"))
        #Output: Hi Mr. Brown

        #Another Example: A function that returns a lambda function
        def mult_by(n):
            return lambda x: x * n

        mult_by_two = mult_by(2)
        print(mult_by_two(6))
        mult_by_five = mult_by(5)
        print(mult_by_five("a"))
        #Output:
```

```
        """
        12
        aaaaa
        """

        #Another lambda function:
        states = ["Hawaii", "Oregon", "California", "Utah", "Colorado"]
        map_length = lambda l: list(zip(l,[len(x) for x in l]))
        print(map_length(states))
        #Output: [('Hawaii', 6), ('Oregon', 6), ('California', 10), ('Utah', 4),
('Colorado', 8)]
```

- Assertions in Python:
  - The `assert` keyword lets you test if a condition in your code returns True or False. If it returns False, a assertion error is thrown. This allows you to debug your code and check it for logical errors.
  - Example:

```
def divide_by_n(x,n):
    assert(n ≠ 0)
    return x/n
divide_by_n(2,0)
```

- Output:

```
AssertionError
Cell In[23], line 1
----> 1 divide_by_n(2,0)

Cell In[22], line 2
      1 def divide_by_n(x,n):
----> 2     assert(n ≠ 0)
      3     return x/n

AssertionError:
```

## PART 4: Handling Files:

- Our main protagonists here are `open()`, `read()` and `write()`
  - Additional parameters for `open()`:
    - `"a"` will append new texts to the end of a file. If the specified file doesn't exist, a new file is created.
    - `"w"` will overwrite existing content. If the specified file doesn't exist, a new file is created.
    - `"x"` will create the specified file. An error is thrown if the file already exists.
  - Example:

```
        f = open("document.txt") #f contains the text inside document.txt
        new_f = open("new_document.txt", "w") #new_f is a new document we made to write
things inside it
        for line in f:
            new_sentence = like_text(line.split())
            new_sentence = " ".join(new_sentence)
            print(new_sentence)
            print(new_sentence, file=new_f)
```

## PART 5: Variable Scope:

- Before we start, global variables are an evil that shouldn't be meddled with, they create unwanted dependencies, but we are gonna learn how to use them just for the sake of learning how to use them
    - Example we will work with:

```
        my_var = "Hey"

        def my_func():
            my_var = "Hello"
            print(my_var)
        my_func()
        print(my_var)
#Output:
"""
Hello
Hey
"""
```

- The global variable remains as it is. Function definitions create a new, local scope for variables. When you assign to a new variable inside a function, the name is only defined within that function. The name of the variable will only be visible inside this function. For this reason, variable names inside and outside of functions can be the same and will not collide.
    - REMINDER: Defining a global variable inside the body of a function introduces dependencies on context. This can limit the reusability of a function and should be avoided!
- Example using global: Use the `global` keyword to create a global variable inside a function (e.g. `global x`).

```
my_var = "Hey"
def my_func():
    global my_var
    my_var = "Hello"
    print(my_var)
my_func()
print(my_var)
#Output:
"""
Hello
Hey
"""
```

- - If you use the `global keyword`, the variable belongs to the global scope.
- ***NOTE:***
    - the global value of lists and dicts can be changed inside a function, without using any keywords or weird things