Functions and lambda expression in python

- A **function** is a block of code which only runs when it is called.
- You can pass data, known as **parameters**, into a function.
- A function can **return** data as a result.
- Information can be passed into functions as **arguments**.

Parameters vs arguments

- A parameter is the variable listed inside the parentheses in the function definition.
- An argument is the value that is sent to the function when it is called.

Arbitrary Arguments, *args

- If the number of arguments is unknown, add a * before the parameter name
- The **positional parameter**(***args**) is **tuple** argument.

```
def my_function(*kids):
 print("The youngest child is " + kids[2])
my_function("Emil", "Tobias", "Linus")
```

```
>> Linus
```

```
Ex2: >>> a,*b=[1,2,3,4]
        >>> a=1
        >>> b=[2, 3, 4]
```

```
a,*b,c=[1,2,3,4]
>>> a=1
>>> b=[2, 3]
>>> c=4
```

Arbitrary Keyword Arguments, **kwargs

- If the number of keyword arguments is **unknown**, add a double ** before the parameter name:
- They used to pass a keyword, **variable-length argument dictionary** to a function.

```
def my_function(**kid):
 print("His last name is " + kid["lname"])
my_function(fname = "Tobias", lname = "Refsnes")
```

```
>> Refsnes
```

Python Import Statements

- lets you import a module into your code.
- A module is a file that contains functions and values that you can reference from your program
- .The import statement syntax is: import modulename

  . **Ex1 : import pandas,numpy , ex2: from random import choice ,**

  With alias : import pandas as pd >> here we use pd instead of pandas

- Python modules are *.py* files that contain Python code.
- Any Python file can be a module. Modules are used to group together related code in a project so you can reuse the same code in different files.

Errors and Exceptions

1.Syntax Errors:

- are mistakes in the source code, such as **spelling** and **punctuation errors**, **incorrect labels**

2. Exceptions

- . Errors detected during execution

```
10 * (1/0)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ZeroDivisionError: division by zero
>>> 4 + spam*3
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'spam' is not defined
>>> '2' + 2
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: can only concatenate str (not "int") to str
```

Handling Exceptions

- the ***try*** *clause* (the statement(s) between the try and except keywords) is executed.
- If no exception occurs, the *except clause* is skipped and execution of the `try` statement is finished.
- If an exception occurs during execution of the try clause, the rest of the clause is skipped. Then, if its type matches the exception named after the except keyword, the *except clause* is executed, and then execution continues after the try/except block.

Python files

- The key function for working with files in Python is the `open()` function.
- `"r"` - Read - Default value. Opens a file for reading, error if the file does not exist
- `"a"` - Append - Opens a file for appending, creates the file if it does not exist
- `"w"` - Write - Opens a file for writing, creates the file if it does not exist
- use the combination of with and open() because the with statement closes the file for you and you get to write less code.
- Method 1:
```
my_file = open("hello.txt", "r")
print(my_file.read())
```
Method 2:
```
with open("hello.txt") as my_file:
    print(my_file.read())
```

Write mode :
```
with open("hello.txt", "w") as my_file:
    my_file.write("Hello world \n")
```