Lesson 4

import os

print("Hello world from ...")

os.system("python --version")

# MODULES

# Every Python program is a module

# Can be called from other files

# A way to break the code in more files

import dog

dog.bark()  # woof!

# from dog import bark

# bark() # No need to call dog because we only imported bark

# This way we pick the specific function we need

# If the other file is in a subfolder, I have to create another file inside the subfolder called \_\_init\_\_.py

from lib import dog2

dog2.bark()  #woof!

# from lib.dog2 import bark

# bark()   #woof!

# In the Python standard library there are a lot pre-built modules

import math

print(math.sqrt(4))  #2.0

# from math import sqrt

# print(sqrt(4))   #2.0

# ACCEPTING ARGUMENTS FROM THE COMMAND LINE

# In Replit: click Schell to open the Command Line. I can clear it and run the python program from there

# Example: python main.py will run this program

# Sometimes you have to type python3

# Terminal in VS

import sys

print(sys.argv)  #If I type python main.py beau 39 in the terminal, it will run main and then print the arguments given, in this case ['main.py', 'beau', '39']

name = sys.argv[1]

print("Hello " + name)  #If I type  python main.py Elettra in the shell, it is going to run the code, print the arguments (line above) ['main.py', 'Elettra'] and print ['main.py', 'Elettra']

# In this way I can give arguments from the shell

import argparse

parser = argparse.ArgumentParser(description="This program prints the name of my dogs")

parser.add\_argument("-c", "--color", metavar="color", required=True, help="the color to search for")

args = parser.parse\_args()

print(args.color) # Typing python main.py -c green will print # ['main.py', 'Elettra']

# If I type only python main.py green, it will say error: the following arguments are required: -c/--color

# parser.add\_argument("-c", "--color", metavar="color", choices={"red", "yellow"}, required=True, help="the color to search for")  # I can only choose from that choices for the argument # If you type a wrong choices, it will tell you which are the choices

# LAMBDA FUNCTIONS

lambda num : num \* 2 # num is the argument, num \* 2 is the expression, in this case it is going to double the argument

lambda a, b : a \* b # Can accept more arguments

# I can assign the function lambda to a variable:

multiply = lambda a, b : a \* b

print(multiply(2, 4))

# The utility of lambda functions comes when combined when other python functionalities, for ex.:

# MAP, FILTER, REDUCE

numbers = [1, 2, 3]

# map():

def double(a):

    return a \* 2

result = map(double, numbers)

print(result)  # Results is a map object, running double for each number # <map object at 0x7f113c5d7fd0>

print(list(result))  # [2, 4, 6]

# Instead with a lambda:

double2 = lambda a : a \* 2

result2 = map(double2, numbers)

print(list(result2))   # [2, 4, 6]

# I can put the lambda directly inside the map:

numbers3 = [4, 5, 6]

result3 = map(lambda a : a \* 2, numbers3)

print(list(result3))   # [8, 10, 12]

# filter()

# filters True values

def isEven(n):

    return n % 2 == 0  # Will return True or False

result4 = filter(isEven, numbers)

print(list(result4))  # [2]

result5 = filter(lambda n : n % 2 == 0, numbers)

print(list(result5))  # [2]

# reduce()

# calculate values out of a list

expenses = [("Dinner", 80), ("Car repair", 120)]  # this is a list of expenses stored as tuples

sum = 0

for expense in expenses:

  sum += expense[1]  #[1] because the price is the item at index 1

print(sum)  # 200

from functools import reduce # you have to import it

sum1 = reduce(lambda a, b : a[1] + b[1], expenses)

# a is the accumulated

# b is the value we are added

# it goes iteratively

print(sum1)  #200

# RECURSION

def factorial(n):

    if n == 1: return 1

    return n \* factorial(n-1)   #It is calling the same function inside the fuction

print(factorial(3))  # 6

print(factorial(4))  # 24

print(factorial(5))  # 120

# DECORATORS

# When you want to change the behaviour of a fuction without modifying the function itself

def logtime(func):

  def wrapper():

    print("before")

    val = func()

    print("after")

    return val

  return wrapper

@logtime   # The function has the logtime decorator assigned

def hello():

  print("hello")

hello()   # before

          # hello

          # after  #logtime takes the function and uses it like an argument

# DOCSTRINGS

# To understand what the code is supposed to do, for others or yourself

# They are different from comments because they follow a certain convenction and they can be implemented automatically

def increment(n):

  """Increment a number"""  # this is what a function does

  return n+1

class Dog:

  """A class representing a dog"""  # this is what the class does

  def \_\_init\_\_(self, name, age):

    """Initialize a new dog"""      # this is what the method does

    self.name = name

    self.age = age

  def bark(self):

    """Let the dog bark"""

    print("WOF!")

# Can be added also at the top of the code:

"""Dog module

This module does ... and provides the following classes:

- Dog

...

"""

# Python will process the docstring and can retrieve them by the help function:

print(help(Dog))

# Help on class Dog in module \_\_main\_\_:

# class Dog(builtins.object)

#  |  Dog(name, age)

#  |

#  |  A class representing a dog

#  |

#  |  Methods defined here:

#  |

#  |  \_\_init\_\_(self, name, age)

#  |      Initialize a new dog

#  |

#  |  bark(self)

#  |      Let the dog bark

#  |

#  |  -----------------------------------------------------------

# -----------

#  |  Data descriptors defined here:

#  |

#  |  \_\_dict\_\_

#  |      dictionary for instance variables (if defined)

#  |

#  |  \_\_weakref\_\_

#  |      list of weak references to the object (if defined)

# ANNOTATIONS

# In Python it is not necessary to specify the type of a variable, but if desired it is possible to do that

def increment4(n: int) -> int:

  return n + 1

# this function receives an int and returns an int

count = 0

count: int = 0

# EXCEPTIONS

# to handle errors

# For ex if you are reading a file, you could get EOFError, which means end of file

try:

 f = 3 + 4 # some lines of code

except EOFError:

 f = 2 + 2 #handler <ERROR1>

except TimeoutError:

 g = 3 #handler <ERROR2>

else:

 h = 7#no exceptions were raised, the code ran successfully

finally:

 g = 4 #do something in any case

print(f, h, g)  # 7 7 4

# k = 2 / 0;

# print(k)  # is gonna give ZeroDivisionError, but I can add it to a try block:

try:

  z = 2 / 0

except ZeroDivisionError:

  print("Cannot divide by zero!")

finally:

  z = 1

print(z)  # Cannot divide by zero!

          # 1

# raise Exception("An error")

# Traceback (most recent call last):

#   File "/home/runner/Lesson-4/main.py", line 266, in <module>

#     raise Exception("An error")

# Exception: An error

# You can raise an error manually to use the try block

try:

  raise Exception("General error")

except Exception as error:

  print(error)   # General error

try:

  raise Exception("General error")

except Exception as error:

  j = 567

  print(j)   # 567

class DogNotFoundException(Exception):

  pass # pass means nothing, only that there are not going to be methods or functions inside the class, there is not going to be any code

try:

    raise DogNotFoundException()

except DogNotFoundException:

  print("Dog not found")  #Dog not found

class DogNotFoundException1(Exception):

  print("ciao")

  pass # pass means nothing, only that there are not going to be methods or functions inside the class, there is not going to be any code

try:

    raise DogNotFoundException1()

except DogNotFoundException1:

  print("Dog ciao")  #ciao

                     #Dog ciao

# with statement

filename = 'C:\\Users\\elettrav\\OneDrive - SINTEF\\Desktop\\Python\_Lesson4.txt'

try:        # we use the try block because it could run into exceptions

  file = open(filename, 'r')

  content = file.read()

  print(content)

finally:

  file.close()   #When you read a file, you need to close it at the end  #Python\_Lesson4

with open(filename, 'r') as file:

  content = file.read()

  print(content)         # with automatically closes it, it is already built in  #Python\_Lesson4