Python Lab Assignment 3

# Program 1: Total cost for setting up a lab

## Introduction

The university is setting up a new lab at their premises. Write a Python function to determine the approximate cost of setting up the lab.

**Total cost = Cost of computers + Cost of chairs + Cost of tables**

If we require 50 computers, chairs and tables, and

Cost of computer = 50'000

Cost of chair = 200

Cost of table = 300

... calculate the total cost.

This program can be done in much simpler ways. However, this is a practice of the concepts of lambda, map and zip...

## Code

**def** labCostCalculator(quantities, costs):

try: totalCost = sum(tuple(map(**lambda** x : x[0] \* x[1], tuple(zip(quantities, costs)))))

except: return "Input error!"

return totalCost

print(labCostCalculator([50, 50, 50], [50000, 200, 300]))

*# If the sizes are mismatched, it will only return the total cost until the last element of the smaller iterable object.*

## Output



# Inferences

Iterable objects enable you to make your code much more succinct, since loops and functions can be used to uniformly operate on their elements in an organized manner.

The lambda keyword enables you quickly and simply simple to complex single-lined functions, as seen here.

# Program 2: PC model data organiser

Write a program that separates the year, PC model and processor generation from a tuple, and store the data in a list. Expected I/P and O/P:

**I/P**: (I7, 2019, Mac, Mac Pro, I5, 2015, I3, I5, I7, Lenovo Yoga, HP Envy, Dell XP, 2021,2020)

**O/P**: {“Year”:[2015, 2019,2020,2021], “Processor”:[I3, I5, I7], “Model”:[ “Mac”, ”Mac Pro”, “Lenovo Yoga”, “HP Envy”, “Dell XP”]}

## Code

from datetime import date

*# Removes extra whitespaces.*

**def** trim(x):

y, j = str(), x[0]

for i in x:

if not i.isspace(): y = y + i

elif not j.isspace(): y = y + " "

j = i

return y[:-1] *# Assumes that '\0' is appended in the function keyAndValue.*

*# Trims out beginning and spaces and brackets*

**def** trimEnds(x):

i = 0

while x[i].isspace(): i = i + 1

x, i = x[i:], 0

while x[i].isspace(): i = i - 1

if i < 0: x = x[:i]

return x

*# Checks whether the give year n is within b years from the current year*

*# (b years to the past or future)*

**def** withinTimeBound(n, b):

currentYear = date.today().year

if n > currentYear - b and n < currentYear + b: return True

return False

*# Checks if an element is a processor generation, year or PC model*

*# Returns the appropriate key and value as a tuple*

**def** keyAndValue(x):

x, i, n = trimEnds(x) + "\0", 0, 0

*# x = x + "\0" is done so that end of string can be easily detected.*

*# n can mean processor name or year, depending on what condition is met*

x, i = x[i:], 0

*# i was the 1st non-whitespace index.*

*# Hence, x loses its initial whitespaces.*

*# This will save some time trimming x, if it turns out to be a model name.*

if x[i] in ("i", "I"):

*# CONDITION 1: Checking if x is a processor generation*

i, n = i + 1, "I"

while x[i].isnumeric(): i, n = i + 1, n + x[i]

if x[i] == "\0" and n != "I": return ("Processor", n)

*# If n == "I", it means no number was found afterwards.*

else:

*# CONDITION 2: Checking of x is a valid year number (unabbreviated)*

while x[i].isnumeric(): i, n = i + 1, 10 \* n + int(x[i])

if x[i] == "\0" and withinTimeBound(n, 1000): return ("Year", n)

*# DEFAULT: Default to PC model*

return ("Model", trim(x))

**def** inputTuple(prompt):

x = trimEnds(input(prompt))

*# Removing possible brackets...*

if x[0] == "(": x = x[1:]

if x[-1] == ")": x = x[:-1]

return tuple(x.split(","))

print("\nPC MODEL DATA ORGANISER\n")

inputData = inputTuple("Input tuple...\n")

outputData = {"Year":[], "Processor":[], "Model":[]}

for x in inputData:

tmp = keyAndValue(x)

outputData[tmp[0]].append(tmp[1])

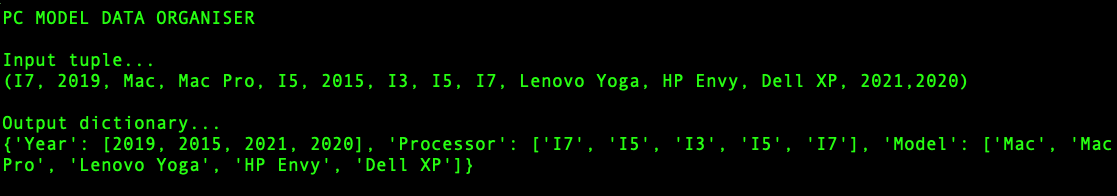
print("\nOutput dictionary...")

print(outputData)

print()

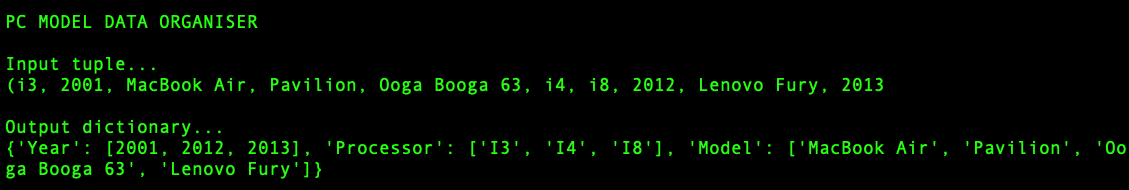
## Outputs

### Using question's data...

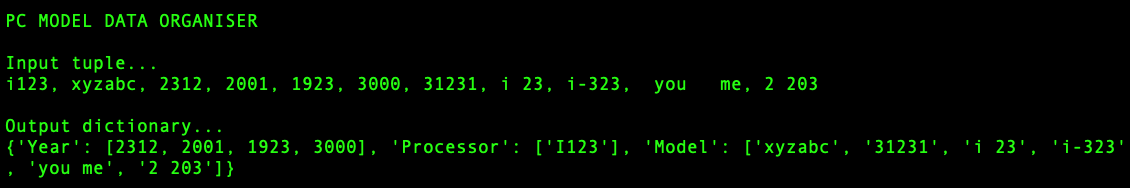


### Using my own data...

#### Normal inputs...



#### Stupid inputs...



# Inferences

The primary advantage of dictionaries is customizable organization of data, as demonstrated here. Furthermore, they uniquely identify each element, since keys are required to be unique.

If you can simplify the job of a called function in a process without impeding your process' efficiency and effectiveness, then do it. In this program, the function "keyAndValue" trims the starting spaces of the received string, so that if it has to be passed to the "trim" function, the "trim" function does not have to go through the same starting spaces again, hence increasing the overall efficiency of the program.

Skipping over spaces and trimming spaces from an input can enhance the flexibility and usability of the program, since the user has to be less concerned about excessive spacing.