```
# tutorial wk 5
# data structures and GUI
# author: B. Schoen-Phelan
# date: 21-10-2020
# some data structures examples from Python 3
# by Dusty Phillips

# class object example
# class Students:
#
#     # pass
#     def __init__(self):
#     # def __init__(self, s_name):
#         print("in students")
#         # self.my_name = s_name
#
#     def say_hello(self):
#         print("hi")
#         print(self.my_name)
#
# students = Students()
# # students = Students('Bianca')
# students.say_hello()

# Tuple example
# stock = ("fb", 75.00, 75.03, 74.90)
# print(stock)
# print(stock[1])
# print(stock[1:3])

# from collections import namedtuple
# Stock = namedtuple("Stock", "symbol current high
low")
# stock = Stock("fb", 75.00, high=75.03, low=74.90)
# print(stock.symbol) # cycle through the different
options
```

```
# Counter class
# from collections import Counter
# def get_letter_freq(sentence):
#     return Counter(sentence)
#
# print(get_letter_freq("hello world"))

stocks={"GOOG":(1253.46, 1258.89, 1241.08),
        "MSFT":(139.76, 140.00, 136.56)}
# print(stocks["GOOG"])
# print(stocks["RIM"])
# print(stocks.get("RIM"))
# print(stocks.get("RIM","not found"))

# print(stocks.keys())
# print(stocks.values())
# print(stocks.items())

# items iterator
# uses hash for efficiency, items are unsorted

# our usual formula does not work
# for s in stocks:
#     print(s.)
# instead:
# for stock, values in stocks.items():
#     print("{} last value is {}".format(stock,
values[0]))

# setting values
# stocks["GOOG"]=(10.00,20.00)
# print(stocks["GOOG"])
# print(stocks)

# different types as keys
random_key_dict = {}
random_key_dict["hello"] = "world"
```

```
random_key_dict[22] = "twenty-two"
random_key_dict[2.2] = "two point two"
random_key_dict[('abc', 123)] = "tuples work"
print(random_key_dict)


class my_dict_obj:
    def __init__(self, v):
        self.my_value = v

my_object = my_dict_obj(3)

random_key_dict[my_object] = "I like objects and
classes"


# for key, value in random_key_dict.items():
#     print("{} has value {}".format(key, value))
#
# try:
#     random_key_dict[[1,2,3]] = "one two three"
# except Exception as e:
#     print(e)

# list sorting behaviours
my_list = ["Hello", "hallo", "HELP", "Helo"]
my_list.sort()
# print(my_list)
# try this - returns None:
my_list = ["Hello", "hallo", "HELP", "Helo"]

# print(my_list.sort())
# my_list = ["Hello", "hallo", "HELP", "Helo"]

# print(sorted(my_list))

my_list.sort(key=str.lower)
```

```
# print(my_list)


# set examples
kitchen_favourites = {"apples", "watermelon",
"strawberries"}
# print(kitchen_favourites)

# for i in kitchen_favourites:
#     print(i)

# print("banana" in kitchen_favourites)
kitchen_favourites.add("carrots")
# print(kitchen_favourites)
kitchen_favourites.add(["pears", "grapes"])
# throws an error

kitchen_favourites.update(["pears", "grapes"])
# use update for adding an iterable
# print(kitchen_favourites)
# print(len(kitchen_favourites))

kitchen_favourites.remove("pears")
# print(kitchen_favourites)

# kitchen_favourites.remove("hello") #will raise an
error, better use try except
# kitchen_favourites.clear()
# print(kitchen_favourites)

# del kitchen_favourites
# print(kitchen_favourites) #will raise an error

other_favourites = {"honey", "maple syrup", "apples",
"pancakes"}
# all_favourites =
other_favourites.union(kitchen_favourites)
```

```
# print(all_favourites)

# kitchen_favourites.update(other_favourites)
# print(kitchen_favourites)

# no order to the outputs, if want ordered
# needs to be pushed into a list
# kitchen_list = list(kitchen_favourites)
# kitchen_list.sort()
# print(kitchen_list)

# other set operations
intersection =
kitchen_favourites.intersection(other_favourites)
# print(intersection)


intersec2 = kitchen_favourites & other_favourites
# print(intersec2)

# items either in one or the other, but not in both
sym_diff =
kitchen_favourites.symmetric_difference(other_favourite
s)
# print(sym_diff)

# print(kitchen_favourites)
# print(other_favourites)
diff = kitchen_favourites.difference(other_favourites)
# print(diff)

diff2= kitchen_favourites - other_favourites
# print(diff2)

diff3 = other_favourites - kitchen_favourites
# print(diff3)
```

```
# queues
# FIFO
from queue import Queue
line = Queue(maxsize=3)
line.put(1)
line.put(2)
line.put(3)

if line.full():
   print("yes, line is full")

# print(line.empty())

line.queue.clear()

# print(line.empty())

# fill with loop
for i in range(1,4):
   line.put(i)

# for i in line: # causes error that queue is not
iterable
#     print(i)

# instead we use while
# while not line.empty():
#     print(line.get()) #get removes them, try just
with line, see output in endless loop
#
# print(line.empty())

# for sorting needs to be cast into list first
line_list = []
for i in range(1,4):
   line_list.append(line.get())
```

```
line_list.sort(reverse=True)
print(line_list)

# move back to queue
for number in line_list:
   line.put(number)

while not line.empty():
   print(line.get())

# import tkinter as tk
# Example 1: Simple Window, nothing in it
# window = tk.Tk()
# window.mainloop()

# Example 2: Window with one Label widget
# window = tk.Tk()
# greeting = tk.Label(text="HELLO WORLD").pack()
# window.mainloop()

# Example 3: Window with frame

# window = tk.Tk()
#
# frame_a = tk.Frame()
# frame_b = tk.Frame()
#
# label_a = tk.Label(master=frame_a, text="I'm in Frame
A")
# label_a.pack()
#
# label_b = tk.Label(master=frame_b, text="I'm in Frame
B")
# label_b.pack()
#
# frame_a.pack()
# frame_b.pack()
```

```
#
# window.mainloop()



# now with proper classes
# also an example of a ttk button
# import tkinter as tk
from tkinter import *

class Tutorial():
    def __init__(self):
        self.create_widgets()

    def create_widgets(self):
        self.root = Tk()

        self.image_lbl = Label(master=self.root)
        image_photo = PhotoImage(file='ailish.gif')
        self.image_lbl.config(image=image_photo)

        self.image_lbl.photo = image_photo
        self.image_lbl.pack()

        self.hi_label = Label(master=self.root)
        self.hi_label["text"] = "Hello World"
        self.hi_label.pack()


        self.hi_there = Button(master=self.root)
        self.hi_there["text"] = "click me!"
        self.hi_there["command"] = self.say_hi
        self.hi_there.pack(side="top")

        self.quit = Button(master=self.root,
text="QUIT", fg="red",
                           command=self.root.destroy)
```

```
        self.quit.pack(side="bottom")
        self.root.mainloop()

    def say_hi(self):
        print("hi there, everyone!")
        self.hi_label.config(text="Hi there, everyone!")
        # self.hi_label.update_idletasks()

        image2 = PhotoImage(file='ailish2.gif')
        self.image_lbl.config(image=image2)
        self.image_lbl.photo = image2

        # self.image_lbl.update_idletasks()
        self.root.update_idletasks()

app = Tutorial()

# The ttk example from the class slides
# from tkinter import *
# from tkinter import ttk
# # example from tkinter docs tutorial
# class Converter:
#     def __init__(self):
#         self.init_window()  # it's good practice to
separate logical functionality
#
#     # this function creates the GUI
#     def init_window(self):
#         root = Tk()  # the window
#         root.title("Feet to Meters")  # title for the
window
#
#         mainframe = ttk.Frame(root)#, padding="3 3 12
12")  # main content frame inside the window
#         mainframe.grid(column=0, row=0, sticky=(N, W,
E, S))  # align to all sides
```

```
#       root.columnconfigure(0, weight=1)  # padding on
resizing
#       root.rowconfigure(0, weight=1)  # padding on
resizing
#
#       self.feet = StringVar()  # Text box, self
allows access from outside this function
#       feet_entry = ttk.Entry(mainframe, width=7,
textvariable=self.feet)  # set fixed size
#       feet_entry.grid(column=2, row=1, sticky=(W, E))
# add to main frame with fixed position
#
#       self.meters = StringVar()  # Text box, self
allows access from outside this function
#       ttk.Label(mainframe, textvariable=self.meters)
\
#           .grid(column=2, row=2, sticky=(W, E))  #
backslash allows break of
#       # very long lines of code
#
#       ttk.Button(mainframe, text="Calculate",
command=self.calculate_conversion) \
#           .grid(column=3, row=3, sticky=W)
#
#       # 3 static labels to explain to the user what
to do here
#       ttk.Label(mainframe,
text="feet").grid(column=3, row=1, sticky=W)
#       ttk.Label(mainframe, text="is equivalent
to").grid(column=1, row=2, sticky=E)
#       ttk.Label(mainframe,
text="meters").grid(column=3, row=2, sticky=W)
#
#       # go through all widgets and adds a bit of
padding to make
#       # the interface look less squashed,
#       # could have been done manually above
```

```
#       for child in mainframe.winfo_children():
#          child.grid_configure(padx=5, pady=5)
#
#       # focus with the cursor goes to the text field
#       feet_entry.focus()
#
#       # pressing return has the same effect as
#       root.bind("<Return>",
self.calculate_conversion)
#
#       # makes everything appear on screen
#       root.mainloop()
#
#    # helper function to calculate the conversion from
feet to meters
#    # takes no arguments
#    # returns no arguments
#    # sets values directly
#    # not great OOP here, but works; we will learn how
to do it
#    # the real OOP way in a few weeks
#    def calculate_conversion(self):
#       try:
#          value = float(self.feet.get())
#          self.meters.set(int(0.3048 * value * 10000.0
+ 0.5) / 10000.0)
#       except ValueError as ve:
#          self.meters.set(ve)
#
#
# # creates an instance of Converter,
# # because of this line the program starts with
# # jumping into __init__
# c = Converter()
```