**Casting in python**

There may be times when you want to specify a type on to a variable. This can be done with casting. Python is an object-orientated language, and as such it uses classes to define data types, including its primitive types.

Casting in python is therefore done using constructor functions:

int() - constructs an integer number from an integer literal, a float literal (by removing all decimals), or a string literal (providing the string represents a whole number)

float() - constructs a float number from an integer literal, a float literal or a string literal (providing the string represents a float or an integer)

str() - constructs a string from a wide variety of data types, including strings, integer literals and float literals

or

we can say that casting refers to changing the data type from one to another

Example of casting

num\_str = "10" # This is a string

num\_int = int(num\_str) # Cast to integer

num\_float = float(num\_str) # Cast to float

Question number 2

List

Lists are used to store multiple items in a single variable.

Lists are one of 4 built-in data types in Python used to store collections of data, the other 3 are [Tuple](https://www.w3schools.com/python/python_tuples.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

Lists are created using square brackets:

### **Example**

Create a List:

thislist = ["apple", "banana", "cherry"]  
print(thislist)

## Access Items

List items are indexed and you can access them by referring to the index number:

### **Example**

Print the second item of the list:

thislist = ["apple", "banana", "cherry"]  
print(thislist[1])

## Loop Through a List

You can loop through the list items by using a for loop:

### **Example**

Print all items in the list, one by one:

thislist = ["apple", "banana", "cherry"]  
for x in thislist:  
  print(x)

## Append Items

To add an item to the end of the list, use the append() method:

### **Example**

Using the append() method to append an item:

thislist = ["apple", "banana", "cherry"]  
thislist.append("orange")  
print(thislist)

## Remove Specified Item

The remove() method removes the specified item.

### **Example**

Remove "banana":

thislist = ["apple", "banana", "cherry"]  
thislist.remove("banana")  
print(thislist)

## Sort List Alphanumerically

List objects have a sort() method that will sort the list alphanumerically, ascending, by default:

### **Example**

Sort the list alphabetically:

thislist = ["orange", "mango", "kiwi", "pineapple", "banana"]  
thislist.sort()  
print(thislist)

Results

['banana', 'kiwi', 'mango', 'orange', 'pineapple']

## Copy a List

You cannot copy a list simply by typing list2 = list1, because: list2 will only be a reference to list1, and changes made in list1 will automatically also be made in list2.

## Use the copy() method

You can use the built-in List method copy() to copy a list.

### **Exampler**

Make a copy of a list with the copy() method:

thislist = ["apple", "banana", "cherry"]  
mylist = thislist.copy()  
print(mylist)

['apple', 'banana', 'cherry']

## List Comprehension

List comprehension offers a shorter syntax when you want to create a new list based on the values of an existing list.

Example:

Based on a list of fruits, you want a new list, containing only the fruits with the letter "a" in the name.

Without list comprehension you will have to write a for statement with a conditional test inside:

### **Example**

fruits = ["apple", "banana", "cherry", "kiwi", "mango"]  
newlist = []  
  
for x in fruits:  
  if "a" in x:  
    newlist.append(x)  
  
print(newlist)

['apple', 'banana', 'mango']

Question no 3

## Tuple

Tuples are used to store multiple items in a single variable.

Tuple is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Set](https://www.w3schools.com/python/python_sets.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A tuple is a collection which is ordered and **unchangeable**.

Tuples are written with round brackets.

### **Example**

Create a Tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple)

## Tuple Items

Tuple items are ordered, unchangeable, and allow duplicate values.

Tuple items are indexed, the first item has index [0], the second item has index [1] etc.

## Ordered

When we say that tuples are ordered, it means that the items have a defined order, and that order will not change.

## Unchangeable

Tuples are unchangeable, meaning that we cannot change, add or remove items after the tuple has been created.

## Allow Duplicates

Since tuples are indexed, they can have items with the same value:

### **Example**

Tuples allow duplicate values:

thistuple = ("apple", "banana", "cherry", "apple", "cherry")  
print(thistuple)

## Access Tuple Items

You can access tuple items by referring to the index number, inside square brackets:

### **Example**

Print the second item in the tuple:

thistuple = ("apple", "banana", "cherry")  
print(thistuple[1])

## Loop Through a Tuple

You can loop through the tuple items by using a for loop.

### **Example**

Iterate through the items and print the values:

thistuple = ("apple", "banana", "cherry")  
for x in thistuple:  
  print(x)

## Join Two Tuples

To join two or more tuples you can use the + operator:

### **Example**

Join two tuples:

tuple1 = ("a", "b", "c")  
tuple2 = (1, 2, 3)  
  
tuple3 = tuple1 + tuple2  
print(tuple3)

Tuples are unchangeable, meaning that you cannot change, add, or remove items once the tuple is created.

But there are some workarounds.

## Change Tuple Values

Once a tuple is created, you cannot change its values. Tuples are **unchangeable**, or **immutable** as it also is called.

But there is a workaround. You can convert the tuple into a list, change the list, and convert the list back into a tuple.

### **Example**

Convert the tuple into a list to be able to change it:

x = ("apple", "banana", "cherry")  
y = list(x)  
y[1] = "kiwi"  
x = tuple(y)  
  
print(x)

## Unpacking a Tuple

When we create a tuple, we normally assign values to it. This is called "packing" a tuple:

### **Example**

Packing a tuple:

## fruits = ("apple", "banana", "cherry") Tuple Methods

Python has two built-in methods that you can use on tuples.

|  |  |
| --- | --- |
| **Method** | **Description** |
| [count()](https://www.w3schools.com/python/ref_tuple_count.asp) | Returns the number of times a specified value occurs in a tuple |
| [index()](https://www.w3schools.com/python/ref_tuple_index.asp) | Searches the tuple for a specified value and returns the position of  where it was found |

myset = {"apple", "banana", "cherry"}

## Set

Sets are used to store multiple items in a single variable.

Set is one of 4 built-in data types in Python used to store collections of data, the other 3 are [List](https://www.w3schools.com/python/python_lists.asp), [Tuple](https://www.w3schools.com/python/python_tuples.asp), and [Dictionary](https://www.w3schools.com/python/python_dictionaries.asp), all with different qualities and usage.

A set is a collection which is unordered, unchangeable\*, and unindexed.

**\* Note:** Set items are unchangeable, but you can remove items and add new items.

Sets are written with curly brackets.

### **Example**

Create a Set:

thisset = {"apple", "banana", "cherry"}  
print(thisset)

## Set Items

Set items are unordered, unchangeable, and do not allow duplicate values.

## Unordered

Unordered means that the items in a set do not have a defined order.

Set items can appear in a different order every time you use them, and cannot be referred to by index or key.

## Unchangeable

Set items are unchangeable, meaning that we cannot change the items after the set has been created.

Once a set is created, you cannot change its items, but you can remove items and add new items.

## Duplicates Not Allowed

Sets cannot have two items with the same value.

### **Example**

Duplicate values will be ignored:

thisset = {"apple", "banana", "cherry", "apple"}  
  
print(thisset)

**Note:** The values True and 1 are considered the same value in sets, and are treated as duplicates:

### **Example**

True and 1 is considered the same value:

thisset = {"apple", "banana", "cherry", True, 1, 2}  
  
print(thisset)

**Note:** The values False and 0 are considered the same value in sets, and are treated as duplicates:

### **Example**

False and 0 is considered the same value:

thisset = {"apple", "banana", "cherry", False, True, 0}  
  
print(thisset)

**Note:** Python does not have built-in support for Arrays, but [Python Lists](https://www.w3schools.com/python/python_lists.asp) can be used instead.

## Arrays

**Note:** This shows you how to use LISTS as ARRAYS, however, to work with arrays in Python you will have to import a library, like the [NumPy library](https://www.w3schools.com/python/numpy/default.asp).

Arrays are used to store multiple values in one single variable:

### **Example**

Create an array containing car names:

## cars = ["Ford", "Volvo", "BMW"] What is an Array?

An array is a special variable, which can hold more than one value at a time.

If you have a list of items (a list of car names, for example), storing the cars in single variables could look like this:

car1 = "Ford"  
car2 = "Volvo"  
car3 = "BMW"

However, what if you want to loop through the cars and find a specific one? And what if you had not 3 cars, but 300?

The solution is an array!

An array can hold many values under a single name, and you can access the values by referring to an index number.

## Access the Elements of an Array

You refer to an array element by referring to the index number.

### **Example**

Get the value of the first array item:

x = cars[0]

### **Example**

Modify the value of the first array item:

cars[0] = "Toyota"

question no 5

shopping\_list = ["milk", "bread", "eggs", "butter", "juice", "sugar", "salt", "biscuits", "tea", "coffee"]

# Display all items

print("Current shopping list:")

for item in shopping\_list:

    print(f"- {item}")

# Add item

add\_item = input("Do you want to add an item? (yes/no): ").lower()

if add\_item == 'yes':

    new\_item = input("Enter item to add: ")

    shopping\_list.append(new\_item)

# Remove item

remove\_item = input("Do you want to remove an item? (yes/no): ").lower()

if remove\_item == 'yes':

    item\_to\_remove = input("Enter item to remove: ")

    if item\_to\_remove in shopping\_list:

        shopping\_list.remove(item\_to\_remove)

    else:

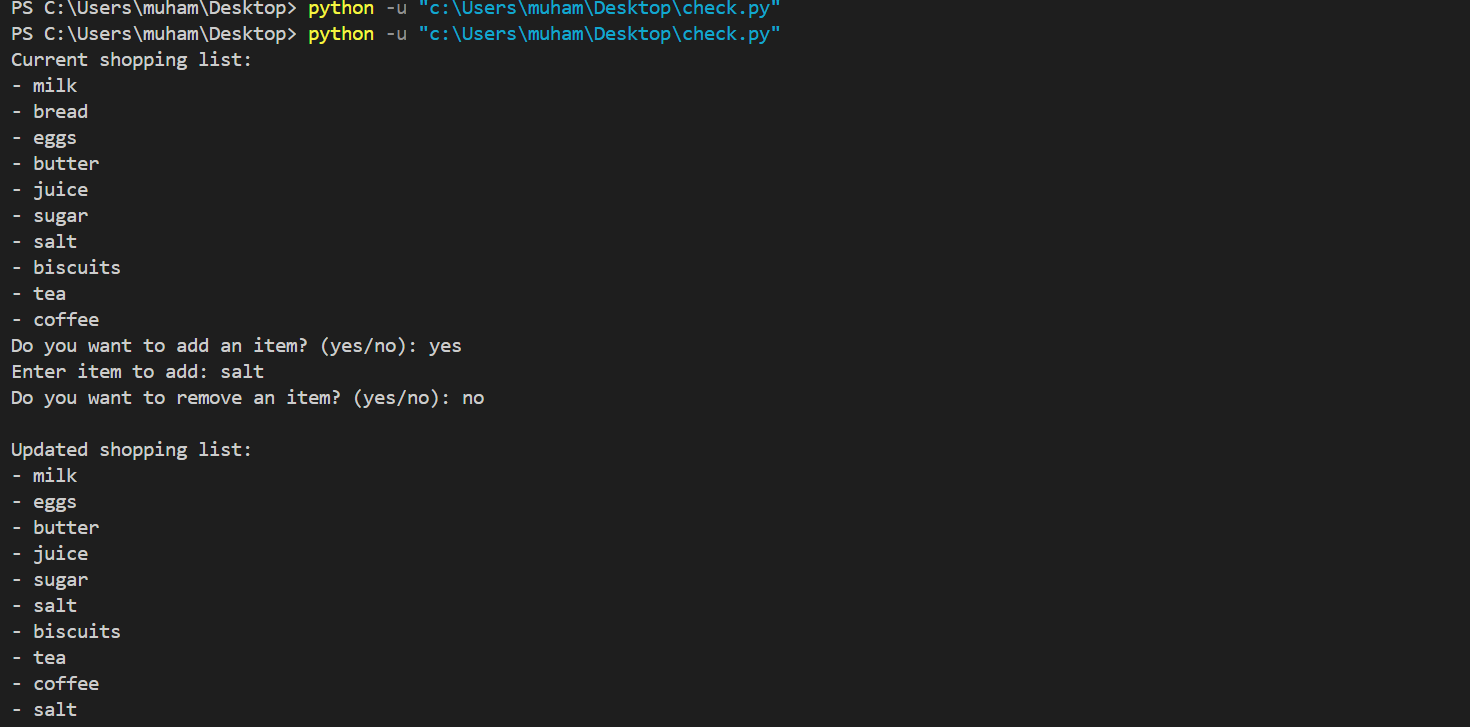
        print("Item not found")

# Display updated list

print("\nUpdated shopping list:")

for item in shopping\_list:

    print(f"- {item}")



Question nu 6

# **Python Functions**

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.

## Creating a Function

In Python a function is defined using the def keyword:

### **Example**

def my\_function():  
  print("Hello from a function")

## Calling a Function

To call a function, use the function name followed by parenthesis:

### **Example**

def my\_function():  
  print("Hello from a function")  
  
**my\_function()**

## Arguments

Information can be passed into functions as arguments.

Arguments are specified after the function name, inside the parentheses. You can add as many arguments as you want, just separate them with a comma.

The following example has a function with one argument (fname). When the function is called, we pass along a first name, which is used inside the function to print the full name:

### **Example**

def my\_function(**fname**):  
  print(fname + " Refsnes")  
  
my\_function(**"Emil"**)  
my\_function(**"Tobias"**)  
my\_function(**"Linus"**)

n = int(input("Enter a number to check if it's even or odd: "))

def check(n):

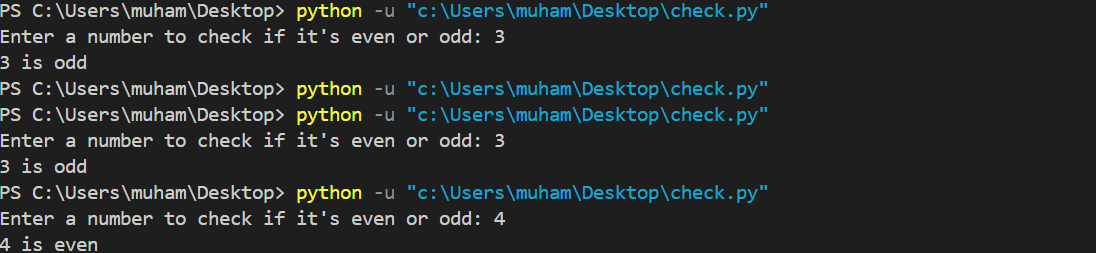
    if n % 2 == 0:

        print(f"{n} is even")

    else:

        print(f"{n} is odd")

check(n)



Question number 7

## What is a Module?

Consider a module to be the same as a code library.

A file containing a set of functions you want to include in your application.

## Create a Module

To create a module just save the code you want in a file with the file extension .py:

### **Example**

Save this code in a file named mymodule.py

def greeting(name):  
  print("Hello, " + name)

## Use a Module

Now we can use the module we just created, by using the import statement:

### **Example**

Import the module named mymodule, and call the greeting function:

import mymodule  
  
mymodule.greeting("Jonathan")

marks = (85, 92, 78, 90, 88)

# Print first and last elements

print("First mark:", marks[0])

print("Last mark:", marks[-1])

#Unpack tuple

m1, m2, m3, m4, m5 = marks

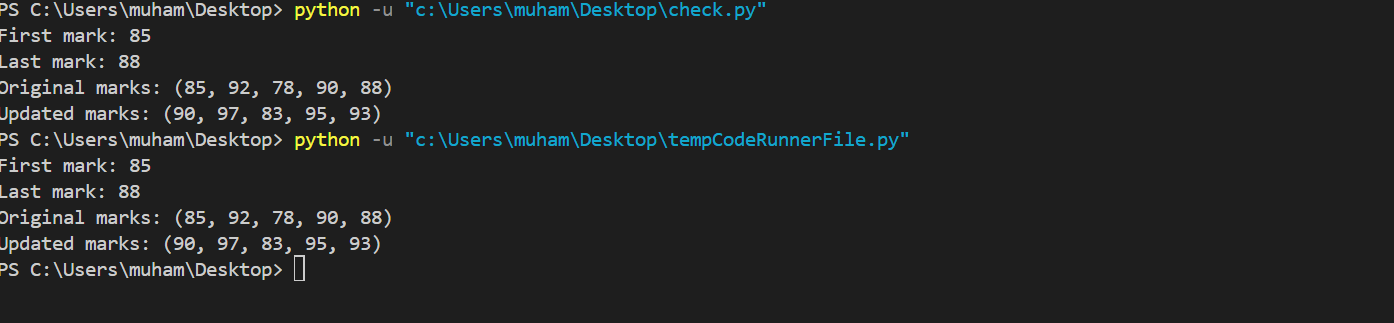
#Create new tuple with +5 marks

new\_marks = tuple(mark + 5 for mark in marks)

# IV. Print both tuples

print("Original marks:", marks)

print("Updated marks:", new\_marks)



Question number 9

odd =0

even =0

divisblebythree = 0

numbers = [5, 12, 7, 18, 9, 24, 3, 16, 11]

for i in range(len(numbers)):

    if numbers[i] % 3 == 0:

        print(f"{numbers[i]} is divisible by 3")

        divisblebythree += 1

        print(f"Total numbers divisible by 3: {divisblebythree}")

    elif numbers[i] % 2 == 0:

        print(f"{numbers[i]} is even and not divisible by 3")

        even += 1

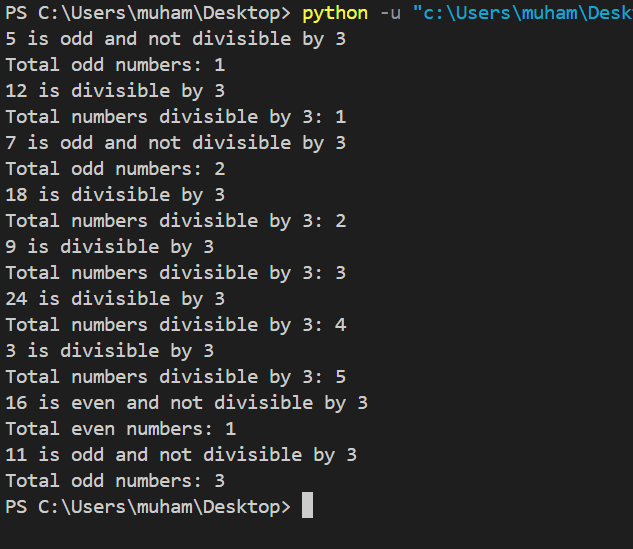
        print(f"Total even numbers: {even}")

    elif numbers[i] % 2 != 0 and numbers[i] % 3 != 0:

        odd += 1

        print(f"{numbers[i]} is odd and not divisible by 3")

        print(f"Total odd numbers: {odd}")



Question number 10

def classify\_numbers(numbers):

    counts = {'positive': 0, 'zero': 0, 'negative': 0}

    for number in numbers:

        if number > 0:

            print(f"{number} is positive")

            counts['positive'] += 1

        elif number == 0:

            print(f"{number} is zero")

            counts['zero'] += 1

        else:

            print(f"{number} is negative")

            counts['negative'] += 1

    return counts

classify\_numbers([-5, 0, 3, 7, -2, 0, 4])

