MCT-242 : COMPUTER PROGRAMMING-I

using Python 3.9

Prepared By:

Mr. Muhammad Ahsan Naeem

YouTube Playlist

https://www.youtube.com/playlist?list=PLWF9TXck7O_wMDB-VriREZ6EvwkWLNB7q

Lab 4: Number Formatters and Complex numbers : CLO 1

Data Type Casting/Conversion:

String Castings:

We have seen casting from int to float and vice versa. This type casting is also possible from string to numerical type and vice versa. For example, consider the following code:

```
x='10'
y=x+5
print(y)
```

As discussed earlier, **`x`** is taken as string and not as number because of the single quotes and hence in line 2, interpreter will generate an error shown below as addition cannot be on between a number and a string.

```
TypeError: can only concatenate str (not "int") to str
```

We can use Type Conversion from String to Int in this case. You might be wondering why will someone have a number stored in a string that later is required to be a number. At this stage, this looks strange but as we will move along in this course you will see that there can be a case that there is important data in a text file in the form of numbers, that file can be read as string and the numbers inside it will be converted to Integers or Floats. Anyhow, let's consider this code:

```
x=int('10')
y=x+5
print(y)
```

or this one:

```
x='10'
y=int(x)+5
print(y)
```

We will not get any error this time and the value of **`y`** will be displayed as 15. Try using **`float()`** instead of **`int()`** and this should work.

Conversion from numeric data type to string is more often desirable than the opposite. Recall from Lab Session 1 where we tried this statement:

```
print('5+5=' + 10)
```

It generates following error:

```
TypeError: can only concatenate str (not "int") to str
```

Now you should read and understand the error. The error name is **TypeError** indicating that there is some issue in data type and the description explains that a string can be concatenated to string only and not the integer. One solution is to use , as concatenation operator instead of + but many times using + is more desirable. Therefore, we can convert numeric value to string and use + for concatenation as shown here:

```
print('5+5=' + str(10))
```

In all conversions programs we created in previous lab sessions, we used , for displaying the results. From now, we will use + operator. A comparison is shown here:

Using , operator:

```
temp=eval(input('Enter temperature in Degrees: '))
f=temp*9/5+32
print(temp,'Degrees = ',f,'Fahrenheit')
```

Using + operator:

```
temp=eval(input('Enter temperature in Degrees: '))
f=temp*9/5+32
print(str(temp)+' Degrees = '+str(f)+' Fahrenheit')
```

Useful Functions in Math Module:

As mentioned in the previous lab session that the whole list of functions available in Math module can be found at:
https://docs.python.org/3/library/math.html
But the most commonly used functions are listed here:

Function	Description
ceil(x)	Returns the smallest integer greater than or equal to x.
copysign(x, y)	Returns x with the sign of y
factorial(x)	Returns the factorial of x
floor(x)	Returns the largest integer less than or equal to x
isfinite(x)	Returns True if x is neither an infinity nor a NaN (Not a Number)
isinf(x)	Returns True if x is a positive or negative infinity
isnan(x)	Returns True if x is a NaN
exp(x)	Returns e**x
log(x[, base])	Returns the logarithm of x to the base (defaults to e)
log2(x)	Returns the base-2 logarithm of x
log10(x)	Returns the base-10 logarithm of x
pow(x, y)	Returns x raised to the power y
sqrt(x)	Returns the square root of x
acos(x)	Returns the arc cosine of x

asin(x)	Returns the arc sine of x
atan(x)	Returns the arc tangent of x
atan2(y, x)	Returns atan(y / x)
cos(x)	Returns the cosine of x
sin(x)	Returns the sine of x
tan(x)	Returns the tangent of x
degrees(x)	Converts angle x from radians to degrees
radians(x)	Converts angle x from degrees to radians
acosh(x)	Returns the inverse hyperbolic cosine of x
asinh(x)	Returns the inverse hyperbolic sine of x
atanh(x)	Returns the inverse hyperbolic tangent of x
cosh(x)	Returns the hyperbolic cosine of x
sinh(x)	Returns the hyperbolic cosine of x
tanh(x)	Returns the hyperbolic tangent of x

Constants in Math Module:

There a few constants defined in Math module given below:

Function	Description
math.pi	The mathematical constant $\pi = 3.141592\ldots$ to available precision.
math.e	The mathematical constant $e = 2.718281\ldots$ to available precision.
math.tau	The mathematical constant $\tau = 6.283185\ldots$ to available precision. Tau is a circle constant equal to 2π, the ratio of a circle's circumference to its radius.
math.inf	A floating-point positive infinity. (For negative infinity, use -math.inf.)
math.nan	A floating-point "not a number" (NaN) value.

So, in one of the tasks, we did in one of previous lab sessions, to find the volume of the sphere we coded like:

```
V=4/3*22/7*r**3
```

This can be done as:

```
import math
V=4/3*math.pi*r**3
```

Multiline Statement:

Sometimes a single statement can be lengthy and Python allows to split that statement into multiple lines using line continuation character \. For example:

```
item_one=3
item_two=4
item_three=5
total=item_one + \
item_two + \
item_three
print(total)
```

A better way to use line continuation is by using indentation as shown:

```
item_one=3
item_two=4
item_three=5
total=item_one + \
     item_two + \
     item_three
print(total)
```

This gives a clear look of the multiline statement.

Tasks:

[1] Write a program that will ask user for the number of seconds and will print out that time in MM:SS format i.e. Minutes:Seconds

Sample Output is:

```
Enter Seconds: 3680
61 : 20
```

[2] Upgrade above task to display the hours also in the format: HH:MM:SS

Sample Output is:

```
Enter Seconds: 7580
2 : 6 : 20
```

Number Formats using f-string:

We have studied the **f-strings** which allows us to insert the placeholder `{}` inside the string and we can write program variable inside those and their value becomes the part of the string. While working on different numbers we might need a specific format to display them. For example, for large numbers we may need to place a comma as **thousand-separator** for better readability as `10,000,000` is better than `10000000`.

We can set different display formats for the number using a `:` inside `{}` and after that specifying the format.

For example, for thousand-separator we can set that as:

```
x=10000000
print(f'{x:,}')
```

And the result will be:

```
10,000,000
```

We can also use the numbers directly as:

```
print(f'{10000000:,}')
```

The following table shows various formatting styles. The formatting code is needed to be place after the colon:

Format	Description	Examples	
		Code	Output
`,`	Use a comma as a thousand separator	`print(f'{1000000:,}')`	`1,000,000`
`_`	Use an underscore as a thousand separator	`print(f'{1000000:_}')`	`1_000_000`

`+`	Use a plus sign to indicate if the result is positive or negative	`print(f'{25:+}')`
		`print(f'{-25:+}')`
			`+25`
			`-25`

`.2f`	To round off floating value to 2 decimal places. (Instead of 2 you can use any other number)	`print(f'{123.3856:.2f}')`	`123.39`
`%`	Percentage format	`print(f'{.512:%}')`	`51.200000%`
`e or E`	Scientific format	`print(f'{123456234.5678:e}')`	`1.234562e+08`
Using Multiple Formatters			
		`print(f'{1000000:+,}')`	`+1,000,000`
		`print(f'{.521678:.2%}')`	`52.17%`
		`print(f'{23456.521676:,.3%}')`	`2,345,652.168%`

		print(f'{123456234.5678:.2e}')	1.23e+08
Number System			

0d	Decimal Format (default format. No need to specify)	print(f'{23}')
		print(f'{23:0d}')
			23
			23

0b	Binary format	print(f'{23:0b}')	10111
0o	Octal Format	print(f'{23:0o}')	27

0x or **0X**	Hexadecimal Format	print(f'{23:0x}')
		print(0b0100)
			17
			4

Padding and Alignment			
>x	Right alignment (Total width of number=x)	print(f'{22:>5}')	22

a>x	Right alignment with padding.
	(Total width of number=x
	Padding with a)
		print(f'{22:0>5}')
		print(f'{123456:0>5}')
		print(f'{23:0>8b}')
			00022
			123456
			00010111

<x	Left alignment (Total width of number=x)	print(f'{22:<5}')	22

a<x	Left alignment with padding.
	(Total width of number=x
	Padding with a)
		print(f'{22:x<5}')
		print(f'{123456:x>5}')
			22xxx
			123456

^x	Center alignment (Total width of number=x)	print(f'{22:^5}')	22

a^x	Center alignment with padding.
	(Total width of number=x
	Padding with a)
		print(f'{22:x^5}')
		print(f'{123456:x^5}')
			x22xx
			123456

For Currency and Time Zones of different countries, there is a Module named locale with details given here: https://docs.python.org/3/library/locale.html			

Using Python built-in round() function:

While working with floating numbers it is quite often desirable to round off the value to some specific number of digits after the decimal point. Python provides a built-in function `round()` for this purpose. The use is shown here:

```
x=10.234
print(round(x))
```

The output will be `10` for this case. So you can see that `round()` rounds off the complete decimal part of the number. You will see that `11` is displayed with this code:

```
x=10.834
print(round(x))
```

It's not always desirable to round off the whole decimal part. We can specify the number of digits in decimal part till which we need to round off as shown here:

```
x=10.234
print(round(x,2))
```

It will round off till 2 decimal points and `10.23` will be displayed.

Tasks:

[3] Upgrade the Task 2 using the appropriate number formatting described in above table so that colon (:) is printed using the formatting and both minutes and seconds are displayed in 2-digits and hours are displayed with 2 or more digits.

Sample Output is:

```
Enter Seconds: 7500
02:05:00
```

Another Sample Output is:

```
Enter Seconds: 360203
100:03:23
```

Complex Numbers:

As we studied earlier, Python supports three types of numbers:

- int
- float
- complex

We have used the integers and floating numbers. For complex numbers, Python uses `j` for the imaginary part of the complex number. Complex numbers can be stored and displayed as shown here:

```
y=3+4j
```

```
print(y)
```

The number is displayed as:

```
(3+4j)
```

Another way to store a complex number is:

```
y=complex(3,4)
print(y)
```

How to input a complex number:

To input a complex number, a function named **`complex()`** is used:

```
x = complex(input("Enter a Complex Number: "))
```

Built-in Mathematical Functions for Complex Numbers:

The basic built-in mathematical functions are applicable to complex numbers as well. For example, below is the code to get two complex numbers from user and display their product:

```
num1 = complex(input("Enter 1st Complex Number (Format : a+bj): "))
num2 = complex(input("Enter 2nd Complex Number (Format : a+bj): "))
num3=num1*num2
print(f"The product is: {num3}")
```

One sample output of above code is:

```
Enter 1st Complex Number (Format : a+bj): 2+3j
Enter 2nd Complex Number (Format : a+bj): 4-5j
The product is: (23+2j)
```

One of the built-in functions in Python is **`abs()`** that gives the absolute value i.e. magnitude of the number e.g. :

```
print(abs(-4))
```

This will display **`4`** on the screen. The same function is applicable on complex numbers as well to find the magnitude of the complex number as shown here:

```
x=4+3j
y=abs(x)
print(y)
```

This will display **`5.0`** which is the magnitude of complex number **`4+3j`**.

Other than the basic functions there are a few functions/methods defined in complex class that are specific for the complex numbers and described here:

- The real part of the complex number can be obtained from the complex number as shown:

```
x=5+4j
a=x.real
```

```
print(a)
```

- Likewise, the imaginary part can be obtained from the complex number as shown:

```
x=5+4j
a=x.imag
print(a)
```

- The conjugate of a complex number can be calculated as:

```
x=5+4j
a=x.conjugate()
print(a)
```

You might have observed a different way of using function in above three examples. This is because complex is a class and a class can have attributes and methods which are used in the way shown above. We will cover the detail of classes in next course of Computer Programming-II

Tasks:

[4] Write a program that will input a complex number and will display the magnitude of that. Do not use **`abs()`** function to find the magnitude; instead, get the real and imaginary part of the entered complex number and calculate the magnitude using this formula:

$$|x| = \sqrt{(Real)^2 + (Imag)^2}$$

Use proper number formatting so that only two digits after decimal points are displayed.

Sample Output is:

```
Enter a Complex Number: 4-5j
The magnitude of entered number is: 6.40
```

[5] Repeat task 4 but this time use the conjugate function of complex class to calculate the magnitude. A complex number when multiplied with its conjugate gives the square of the magnitude of that complex number.

Note→ *While attempting to find the square-root of the product of the number and the conjugate, you will get an error as* **`sqrt()`** *function cannot be applied to a complex number. The imaginary part of the product is 0 but still it is a complex number. So use the real part of the product in* **`sqrt()`** *function.*

[6] Write a program that will take input a vector (in form of complex number) from user and will display the unit vector (in form of complex number) of that.

Use proper number formatting so that only two digits after decimal points are displayed.

Sample output is:

```
Enter a Vector (in the form of Complex Number): 2-3j
The Unit vector of entered vector is:
(0.55-0.83j)
```