Introduction to Python

Applications of python

- Web development
- Data science
- Machine learning
- Automation

First program

```
print()

print("Hello world")

print(1)

print(3.21)
```

Datatypes

- Text - String – "Hello world" or 'Hello world'
- Number
    - Integer – whole number: positive/negative number without decimal point
    - Float - positive/negative number with one or more decimals

Arithmetic Operations

```
//convert days to minutes

print(20*24*60)

//to make it understandable

print( "20 days are")
print(20*24*60)
print("minutes")
//to bring two prints in the same line: string concatenation

print("20 days are"+ 28800 + "minutes")

        # + sign is used for string concatenation

        # 28800 is in integer format in between two strings

        # typecast is done to convert the data type

print("20 days are"+ str(28800) + "minutes")

        #Alternate method

print(f"20 days are {28800} minutes")
```

```
        #number to printed is not typecasted instead included within {}

        #math operations can be included directly

print(f"20 days are {20*24*60} minutes")

        #for multiple calculations

print(f"56 days are {56*24*60} minutes")
print(f"88 days are {88*24*60} minutes")
print(f"100 days are {100*24*60} minutes")
                # in the above example most of the content are common

                    print(f"56 days are {56*24*60} minutes")
                    print(f"88 days are {88*24*60} minutes")
                    print(f"100 days are {100*24*60} minutes")
                # Common contents can be put inside a variable

                # Variables: memory locations/storage containers for storing values

// Variables

calculation_to_seconds = 24*60*60       #datatype is dynamically assigned to the variables

        # variable names cannot contain the following reserved words
```

and	except	lambda	with
as	finally	nonlocal	while
assert	false	None	yield
break	for	not	
class	from	or	
continue	global	pass	
def	if	raise	
del	import	return	
elif	in	True	
else	is	try	

```
print(f"56 days are {56* calculation_to_seconds } ~~minutes~~ seconds")
print(f"88 days are {88* calculation_to_seconds } minutes")
print(f"100 days are {100* calculation_to_seconds } minutes")
        #units are also changed similarly like the calculation so, it can also be declared as a variable

calculation_to_seconds = 24*60*60
name_of_unit = "seconds"
print(f"56 days are {56* calculation_to_seconds } {name_of_unit }")
print(f"88 days are {88* calculation_to_seconds } {name_of_unit }")
print(f"100 days are {100* calculation_to_seconds } {name_of_unit }")
        # name_of_unit is a variable and not a string, so it is included within {}

        # advantage of having variables
```

```
calculation_to_units = 24
name_of_unit = “hours”
print(f“56 days are {56* calculation_to_seconds } {name_of_unit }”)
print(f“88 days are {88* calculation_to_seconds } {name_of_unit }”)
print(f“100 days are {100* calculation_to_seconds } {name_of_unit }”)
//Functions

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units():   #function definition
        print(f“56 days are {56* calculation_to_seconds } {name_of_unit }”)

days_to_units()    #function call


//Function parameters

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        print(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of_unit}”)

days_to_units(56)    #function call
days_to_units(88)
days_to_units(100)


//Function parameters – multiple

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days, custom_message):   #function definition
        print(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of_unit}”)
        print(custom_message)

days_to_units(56,”Awesome!”)    #function call
days_to_units(88,”Good!”)
```

```
//Scope of variables

Global variables: calculation_to_units, name_of_unit

        # can be accessed anywhere within the program

Local variables: num_of_days, custom_message

        # can be accessed only within days_to_units() function

//User input

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        print(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)

user_input = input(“Enter the number of days:\n”)
        #input() returns a value entered by the user which can be assigned to a variable
//Function returning a value

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)

user_input = input(“Enter the number of days:\n”)  #getting input
calculated_value = days_to_units(user_input)  #function call and return
print(calculated_value)
        ## Gives unexpected results
calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)

user_input = input(“Enter the number of days:\n”)
user_input_number = int(user_input)  # type casting
calculated_value = days_to_units(user_input_number)
print(calculated_value)
```

//Conditional statements

in the previous program, the user input is not checked before passing it into the function – no validation, eg: try entering 0,-10,"reht". For the first two case, we may get a illogical output and the third case would crash the program

```
calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        if(num_of_days>0):
                return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")

user_input = input("Enter the number of days:\n")
user_input_number = int(user_input)  # type casting
calculated_value = days_to_units(user_input_number)
print(calculated_value)
####

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        if(num_of_days>0):
                return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")
        else:
                return("You have entered a negative number")
user_input = input("Enter the number of days:\n")
user_input_number = int(user_input)  # type casting
calculated_value = days_to_units(user_input_number)
print(calculated_value)
# comparison operators: >, <, >=, <=, ==, !=

# arithmetic operators: +, -, *, /, %, **, //

# result of a condition: true or false → Boolean data type. Try print(num_of_days>0)

# type() can be used for checking the data type of a variable. Eg: type(num_of_days>0) would return a data type called 'bool'. Try for other data types.

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        if(num_of_days>0):
```

```
            return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")
        elif num_of_days==0:
            return("You have entered a 0")
        else:
            return("You have entered a negative number")
user_input = input("Enter the number of days:\n")
user_input_number = int(user_input)  # type casting
calculated_value = days_to_units(user_input_number)
print(calculated_value)
# above program works for user input only if the data type is integer. Other data type inputs would crash the program.

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        if(num_of_days>0):
            return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")
        elif num_of_days==0:
            return("You have entered a 0")
        else:
            return("You have entered a negative number")
user_input = input("Enter the number of days:\n")
if user_input.isdigit(): #isdigit() returns true if the string is a digit string. Input is passed using . operator
        user_input_number = int(user_input)  # type casting
        calculated_value = days_to_units(user_input_number)
        print(calculated_value)
else:
        print("Input is not valid")
# try entering 0, negative number, float and string

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        if(num_of_days>0):
            return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")
        elif num_of_days==0:
            return("You have entered a 0")
        else:
            return("You have entered a negative number")
```

```
def validate_and_execute():
        if user_input.isdigit(): #isdigit() returns true if the string is a digit string. Input is passed using . operator
                user_input_number = int(user_input)  # type casting
                calculated_value = days_to_units(user_input_number)
                print(calculated_value)
        else:
                print("Input is not valid")

user_input = input("Enter the number of days:\n")
validate_and_execute()
# try entering 0, negative number, float and string
```

//Nested Conditional statements

```
calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")


def validate_and_execute():
        if user_input.isdigit():
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
        else:
                print("Input is not valid")

user_input = input("Enter the number of days:\n")
validate_and_execute()
# try entering 0, negative number, float and string
```

// Try and except

```
calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")
```

```
def validate_and_execute():
        try:
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
        except ValueError:    #ValueError can be removed to catch any type of errors
                print("Input is not valid")

user_input = input("Enter the number of days:\n")
validate_and_execute()
# try entering 0, negative number, float and string

#negative number will not create any error. So except block might miss it

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")



def validate_and_execute():
        try:
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
                else user_input_number <0:
                        return("You entered a negative number")
        except ValueError:    #ValueError can be removed to catch any type of errors
                print("Input is not valid")

user_input = input("Enter the number of days:\n")
validate_and_execute()
```

// While Loops

```
calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)


def validate_and_execute():
        try:
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
                else user_input_number <0:
                        return(“You entered a negative number”)
        except:
                print(“Input is not valid”)

while True:  # run indefinitely. True is boolean
        user_input = input(“Enter the number of days:\n”)
        validate_and_execute()
# try entering 0, negative number, float and string

#user can exit the program

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)


def validate_and_execute():
        try:
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
```

```
                else user_input_number <0:
                        return("You entered a negative number")
        except:
                print("Input is not valid")

while user_input != "exit":   #the variable is not used before while. It will create warning
        user_input = input("Enter the number of days:\n")
        validate_and_execute()
####

calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")


def validate_and_execute():
        try:
                user_input_number = int(user_input)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
                else user_input_number <0:
                        return("You entered a negative number")
        except:
                print("Input is not valid")

user_input = ""
while user_input != "exit":
        user_input = input("Enter the number of days:\n")
        validate_and_execute()
```

// For Loops and List

#List – data type that has data within [] separated by comma

above program is not configured for accepting list input. Ie: user cannot enter all his required values at once.

above program can be modified to accept all the user's input at once and validate it individually

```
calculation_to_units = 24
name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f"{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}")


def validate_and_execute():
        try:
                user_input_number = int(num_of_days_element)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
                else user_input_number <0:
                        return("You entered a negative number")
        except:
                print("Input is not valid")

user_input = ""
while user_input != "exit":
        user_input = input("Enter the number of days:\n")
        for num_of_days_element in user_input:      # num_of_days_element represents each element in the list as str
                validate_and_execute()
        for num_of_days_element in user_input.split():   #.split() takes the input that are space separated.
                                                                Eg: 10 23 25
                validate_and_execute()
         for num_of_days_element in user_input.split(","):   #.split() takes the input that are comma
                                                                separated. Eg: 10,23,25
                validate_and_execute()
```

try input: 10,20,"aeiou",95.36
try type(user_input.split(",")) → observe the type – list and each element is string

```
calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)


def validate_and_execute():
        try:
                user_input_number = int(num_of_days_element)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
                else user_input_number <0:
                        return(“You entered a negative number”)
        except:
                print(“Input is not valid”)

user_input = ””
while user_input != “exit”:
        user_input = input(“Enter the number of days as comma separated values:\n”)
        for num_of_days_element in user_input.split(“,”):      # for inputs 10,12,45,89
                validate_and_execute()
        for num_of_days_element in user_input.split(“, ”):   # for inputs 10, 12, 45, 89    ###space
                validate_and_execute()

//List Operations
my_list = [“January”, “February”, “March”]
#elements within list are indexed and starts with index 0
#to access an element:
Eg: my_list[0] points to “January”

my_list = [“January”, “February”, “March”]
print(my_list[1])              ## prints February

my_list.append(“April”)
print(my_list)              ## list contains 4 elements

my_list.remove(“January”)
print(my_list)
# for single line comment “”” (triple quotes) at the start and end for block comment
```

```
my_list = [“January”, “February”, “March”, “January”]
my_list.remove(“January”)
print(my_list)
# first occurrence will be removed

//List Operations – Extended
myList = [1, 2, 3, 'ABC', 'xyzw']

# Insert single element at the end
myList.append(4)
myList.append(5)
myList.append(6)
for i in range(7, 9):
   myList.append(i)
print(myList)

output: [1, 2, 3, 'ABC', 'xyzw', 4, 5, 6, 7, 8]

# Insert multiple element at the end
myList.extend([4, 5, 6])
for i in range(7, 9):
   myList.append(i)
print(myList)

output: [1, 2, 3, 'ABC', 'xyzw', 4, 5, 6, 7, 8]

# Insert single element in desired position
myList.insert(3, 10)
myList.insert(4, 85)
myList.insert(5, 76)
print(myList)

output: [1, 2, 3, 10, 85, 76, 'ABC', 'xyzw']

# Remove an element from desired position
myList.pop(4)
myList.insert(4, 'pyhton')
myList.insert(5, 'language')
myList.insert(6, 'so much fun!')
print(myList)

output: [1, 2, 3, 'ABC', 'pyhton', 'language', 'so much fun!']

# Print section of a list
print(myList[:4])  # prints from beginning to end index
```

```
## output: [1, 2, 3, 'ABC']
print(myList[2:])  # prints from start index to end of list
## output: [3, 'ABC', 'xyzw']
print(myList[2:4]) # prints from start index to end index
## output: [3, 'ABC']
print(myList[:])   # prints from beginning to end of list
## output: [1, 2, 3, 'ABC', 'xyzw']

# Reverse the List entries
print(myList[::-1])  # does not modify the original list
## output: ['xyzw', 'ABC', 3, 2, 1]
myList.reverse()    # modifies the original list
print(myList)
## output: ['xyzw', 'ABC', 3, 2, 1]

# Length of the list
print(len(myList))
##output: 5

# Minimum and Maximum values
print(min([1, 2, 3]))
print(max([1, 2, 3]))
##output: 1
3

# Number of occurrences of a given element in the list
print(myList.count(3))
##output: 1

# concatenate
yourList = [4, 5, 'Python', 'is fun!']
print(myList+yourList)
##output: [1, 2, 3, 'ABC', 'xyzw', 4, 5, 'Python', 'is fun!']

# Multiply
print(myList+yourList)
##output: [1, 2, 3, 'ABC', 'xyzw', 1, 2, 3, 'ABC', 'xyzw']

# Sort
yourList = [4, 2, 6, 5, 0, 1]
yourList.sort()
print(yourList)
##output: [0, 1, 2, 4, 5, 6]
```

```
// Set

# same as list but does not allow duplicate entries
# list can be converted to set()
# upon conversion, duplicate values are removed

calculation_to_units = 24
name_of_unit = “hours”

        ##two blank lines are needed before a function
def days_to_units(num_of_days):   #function definition
        return(f“{num_of_days} days are {num_of_days * calculation_to_seconds} {name_of _unit}”)


def validate_and_execute():
        try:
                user_input_number = int(num_of_days_element)  # type casting
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number)
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
                else user_input_number <0:
                        return(“You entered a negative number”)
        except:
                print(“Input is not valid”)

user_input = ””
while user_input != “exit”:
        user_input = input(“Enter the number of days as comma separated val ues:\n”)
        list_of_days = user_input.split(“, ”)
        print(list_of_days)
        print(set(list_of_days))
        print(type(list_of_days))
        print(type(set(list_of_days)))   # 1. Set() 2. Type() 3. Print()
        for num_of_days_element in set(user_input.split(“, ”)):
                validate_and_execute()

# try 10, 47, 55, 10
# check the list and set type. Check for duplicates
# list is represented by [] whereas set is represented by {}

# set creation
my_set = {“January”, “February”, “March”}
# elements can be added and removed
```

elements can be accessed only using for loop, individual element access is prohibited
elements in a set do not have a defined order
elements can be added or removed but cannot be changed

```
for element in my_set:
        print(element)
```

#element addition
my_set.add("April")
print(my_set)
#observe the order of elements

#element deletion
my_set.remove("January")
print(my_set)

//Built in Functions
print() → prints to standard output device
input() → asks for user's input and assigns to a variable as string
set() → takes the input as list and converts into a set
int() → takes string as input and converts to integer
other built-in functions:

abs()	delattr()	hash()	memoryview()	set()
all()	dict()	help()	min()	setattr()
any()	dir()	hex()	next()	slice()
ascii()	divmod()	id()	object()	sorted()
bin()	enumerate()	input()	oct()	staticmethod()
bool()	eval()	int()	open()	str()
breakpoint()	exec()	isinstance()	ord()	sum()
bytearray()	filter()	issubclass()	pow()	super()
bytes()	float()	iter()	print()	tuple()
callable()	format()	len()	property()	type()
chr()	frozenset()	list()	range()	vars()
classmethod()	getattr()	locals()	repr()	zip()
compile()	globals()	map()	reversed()	__import__()
complex()	hasattr()	max()	round()	

String_var.split()
String_var.isdigit()
# Above functions are for string data type
# In python, there are such functions for each datatype
# eg: type a list: [10, 2, 5] and put a dot, pycharm would display all functions pertaining to the list data type

// Dictionary

In the pervious example problem of calculating hours for specific number of days, it can be modified to receive the number of days as well as the unit for conversion from the user
Instead of list entry the user now needs to enter a single value with conversion unit that are ":" separated. Separation operator is customizable.
Dictionary datatype contains data in key: value pairs
syntax: {"days": 20, "unit": "hours"}
"days" and "unit" are called keys
20 and "hours" are called values

```
        .
        .
        .
        ##Accepting user input

user_input = ""
while user_input != "exit":
        user_input = input("Enter the number of days as comma separated values:\n")
        days_and_unit = user_input.split(":")  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {"days": days_and_unit[0], "unit": days_and_unit[1]}
        # print the list and dictionary separately to verify

####


#calculation_to_units = 24
#name_of_unit = "hours"

        ##two blank lines are needed before a function
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit     == "hours":
                return(f"{num_of_days} days are {num_of_days * 24} hours")
        elif conversion_unit     == "minutes":
                return(f"{num_of_days} days are {num_of_days * 24*60} minutes")
        elif conversion_unit     == "seconds":
                return(f"{num_of_days} days are {num_of_days * 24*60*60} seconds")
        else:
                return("Unsupported unit")



def validate_and_execute():
        try:
```

```
                user_input_number = int(days_and_unit_dictionary[“days”])  # indexed using key
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number, days_and_unit_dictionary[“unit”])
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
                else user_input_number <0:
                        return(“You entered a negative number”)
        except:
                print(“Input is not valid”)

user_input = ””
while user_input != “exit”:
        user_input = input(“Enter the number of days as comma separated values:\n”)
        days_and_unit = user_input.split(“:”)  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {“days”: days_and_unit[0], “unit”: days_and_unit[1]}
        validate_and_execute()

//Modules
#1 segmenting the code
## main.py
user_input = ””
while user_input != “exit”:
        user_input = input(“Enter the number of days as comma separated values:\n”)
        days_and_unit = user_input.split(“:”)  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {“days”: days_and_unit[0], “unit”: days_and_unit[1]}
        validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit      == “hours”:
                return(f“{num_of_days} days are {num_of_days * 24} hours”)
        elif conversion_unit      == “minutes”:
                return(f“{num_of_days} days are {num_of_days * 24*60} minutes”)
        elif conversion_unit      == “seconds”:
                return(f“{num_of_days} days are {num_of_days * 24*60*60} seconds”)
        else:
                return(“Unsupported unit”)



def validate_and_execute():
        try:
                user_input_number = int(days_and_unit_dictionary[“days”])
```

```
            if user_input_number > 0:
                calculated_value = days_to_units(user_input_number, days_and_unit_dictionary[“unit”])
                print(calculated_value)
            elif user_input_number == 0:
                return(“You have entered a 0”)
            else user_input_number <0:
                return(“You entered a negative number”)
        except:
            print(“Input is not valid”)

#2 Need to resolve the referencing issues
## main.py
import helper            # import the entire the file and use reference to that file
user_input = ””
while user_input != “exit”:
        user_input = input(“Enter the number of days as comma separated values:\n”)
        days_and_unit = user_input.split(“:”)  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {“days”: days_and_unit[0], “unit”: days_and_unit[1]}
        helper.validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit     == “hours”:
            return(f“{num_of_days} days are {num_of_days * 24} hours”)
        elif conversion_unit     == “minutes”:
            return(f“{num_of_days} days are {num_of_days * 24*60} minutes”)
        elif conversion_unit     == “seconds”:
            return(f“{num_of_days} days are {num_of_days * 24*60*60} seconds”)
        else:
            return(“Unsupported unit”)


def validate_and_execute(days_and_unit_dictionary):
        try:
            user_input_number = int(days_and_unit_dictionary[“days”])
            if user_input_number > 0:
                calculated_value = days_to_units(user_input_number, days_and_unit_dictionary[“unit”])
                print(calculated_value)
            elif user_input_number == 0:
                return(“You have entered a 0”)
            else user_input_number <0:
                return(“You entered a negative number”)
        except:
            print(“Input is not valid”)
```

```
#3 Alternate method for referencing
## main.py
import helper as h              # import the entire the file and use reference to that file
user_input = ""
while user_input != "exit":
        user_input = input("Enter the number of days as comma separated values:\n")
        days_and_unit = user_input.split(":")  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {"days": days_and_unit[0], "unit": days_and_unit[1]}
        h.validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit     == "hours":
                return(f"{num_of_days} days are {num_of_days * 24} hours")
        elif conversion_unit     == "minutes":
                return(f"{num_of_days} days are {num_of_days * 24*60} minutes")
        elif conversion_unit     == "seconds":
                return(f"{num_of_days} days are {num_of_days * 24*60*60} seconds")
        else:
                return("Unsupported unit")


def validate_and_execute(days_and_unit_dictionary):
        try:
                user_input_number = int(days_and_unit_dictionary["days"])
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number, days_and_unit_dictionary["unit"])
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
                else user_input_number <0:
                        return("You entered a negative number")
        except:
                print("Input is not valid")


#4 Alternate method for specific import
## main.py
from helper import validate_and_execute    # import a specific function from the file
user_input = ""
while user_input != "exit":
        user_input = input("Enter the number of days as comma separated values:\n")
        days_and_unit = user_input.split(":")  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {"days": days_and_unit[0], "unit": days_and_unit[1]}
```

```
        validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit     == “hours”:
                return(f“{num_of_days} days are {num_of_days * 24} hours”)
        elif conversion_unit     == “minutes”:
                return(f“{num_of_days} days are {num_of_days * 24*60} minutes”)
        elif conversion_unit     == “seconds”:
                return(f“{num_of_days} days are {num_of_days * 24*60*60} seconds”)
        else:
                return(“Unsupported unit”)


def validate_and_execute(days_and_unit_dictionary):
        try:
                user_input_number = int(days_and_unit_dictionary[“days”])
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number, days_and_unit_dictionary[“unit”])
                        print(calculated_value)
                elif user_input_number == 0:
                        return(“You have entered a 0”)
                else user_input_number <0:
                        return(“You entered a negative number”)
        except:
                print(“Input is not valid”)

#5 Import variables
## main.py
from helper import validate_and_execute, user input message
user_input = ””
while user_input != “exit”:
        user_input = input (user input message)
        days_and_unit = user_input.split(“:”)  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {“days”: days_and_unit[0], “unit”: days_and_unit[1]}
        validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit     == “hours”:
                return(f“{num_of_days} days are {num_of_days * 24} hours”)
        elif conversion_unit     == “minutes”:
                return(f“{num_of_days} days are {num_of_days * 24*60} minutes”)
        elif conversion_unit     == “seconds”:
                return(f“{num_of_days} days are {num_of_days * 24*60*60} seconds”)
        else:
```

```
                return("Unsupported unit")


def validate_and_execute(days_and_unit_dictionary):
        try:
                user_input_number = int(days_and_unit_dictionary["days"])
                if user_input_number > 0:
                        calculated_value = days_to_units(user_input_number, days_and_unit_dictionary["unit"])
                        print(calculated_value)
                elif user_input_number == 0:
                        return("You have entered a 0")
                else user_input_number <0:
                        return("You entered a negative number")
        except:
                print("Input is not valid")

        user input message = "Enter the number of days as comma separated values:\n"

#6 Import Everything
## main.py
from helper import *    # imports everything from the helper module. But this import does not need
reference to the helper module while calling function or variables like mentioned in #2
user_input = ""
while user_input != "exit":
        user_input = input (user input message)
        days_and_unit = user_input.split(":")  # to accept colon separated input
        # conversion of list to dictionary by accessing individual elements
        days_and_unit_dictionary = {"days": days_and_unit[0], "unit": days_and_unit[1]}
        validate_and_execute(days_and_unit_dictionary)
## helper.py
def days_to_units(num_of_days, conversion_unit):   #function definition
        if conversion_unit      == "hours":
                return(f"{num_of_days} days are {num_of_days * 24} hours")
        elif conversion_unit      == "minutes":
                return(f"{num_of_days} days are {num_of_days * 24*60} minutes")
        elif conversion_unit      == "seconds":
                return(f"{num_of_days} days are {num_of_days * 24*60*60} seconds")
        else:
                return("Unsupported unit")


def validate_and_execute(days_and_unit_dictionary):
        try:
                user_input_number = int(days_and_unit_dictionary["days"])
                if user_input_number > 0:
```

```
                calculated_value = days_to_units(user_input_number, days_and_unit_dictionary["unit"])
                print(calculated_value)
        elif user_input_number == 0:
                return("You have entered a 0")
        else user_input_number <0:
                return("You entered a negative number")
except:
        print("Input is not valid")

user input message = "Enter the number of days as comma separated values:\n"
```

// Built-in modules of python - Date and Time

Five main object classes in this module

1. datetime.date : It allows us to manipulate date without interfering time (month, day, year)
2. datetime.time : It allows us to manipulate date without interfering date (hour, minute, second, microsecond)
3. datetime.datetime : It allows us to manipulate the combination of date and time (month, day, year, hour, second, microsecond).
4. datetime.tzinfo : An abstract class for dealing with time zones. These types of objects are immutable. For instance, to account for different time zones and/or daylight saving times.
5. datetime.timedelta : It is the difference between two date, time or datetime instances; the resolution ca

```
# Creating DateTime Objects
# import datetime class from datetime module

from datetime import datetime
# get today's date
today = datetime.now()
print(today)
dt_nw = datetime.now()
# to get hour from datetime
print('Hour: ', dt_nw.hour)
# to get minute from datetime
print('Minute: ', dt_nw.minute)
```

output:

```
2022-12-12 09:49:52.357742
Hour:  9
Minute:  49
```

```
# Timespan and Time Differences
#import timedelta
from datetime import timedelta, datetime
```

```
print(timedelta(days= 365, hours= 12, minutes= 30))
# get current time
now = datetime.now()
print ("Today's date & time: ", str(now))

#add 365 days to current date
future_date_after_one_year = now + timedelta(days = 365)
print('Date & time after one year: ', future_date_after_one_year)

#subtract 7 days from current date
seven_days_ago = now - timedelta(days = 7)
print('Date & time seven days ago: ', seven_days_ago)
# print('seven_days_ago object type: ', type(seven_days_ago))
```

Output:

```
365 days, 12:30:00
Today's date & time:  2022-12-12 09:52:39.351678
Date & time after one year:  2023-12-12 09:52:39.351678
Date & time seven days ago:  2022-12-05 09:52:39.351678
```

```
# Date Formatting
## strptime() can read strings with date and time information and convert them to datetime objects
# import datetime
from datetime import datetime
date_str = "2 january, 2020"
# format date
date_obj = datetime.strptime(date_str, "%d %B, %Y")
print("Today's date is: ", date_obj)
```

output:

```
Today's date is:  2020-01-02 00:00:00
```

```
## strftime() converts datetime objects back into strings
# current date and time
now = datetime.now()
# format time in HH:MM:SS
time = now.strftime("%H:%M:%S")
print("Time:", time)
# format date
date_time = now.strftime("%m/%d/%Y, %H:%M:%S")
print("Date and Time:",date_time)
```

output:

```
Time: 09:54:02
Date and Time: 12/12/2022, 09:54:02
```

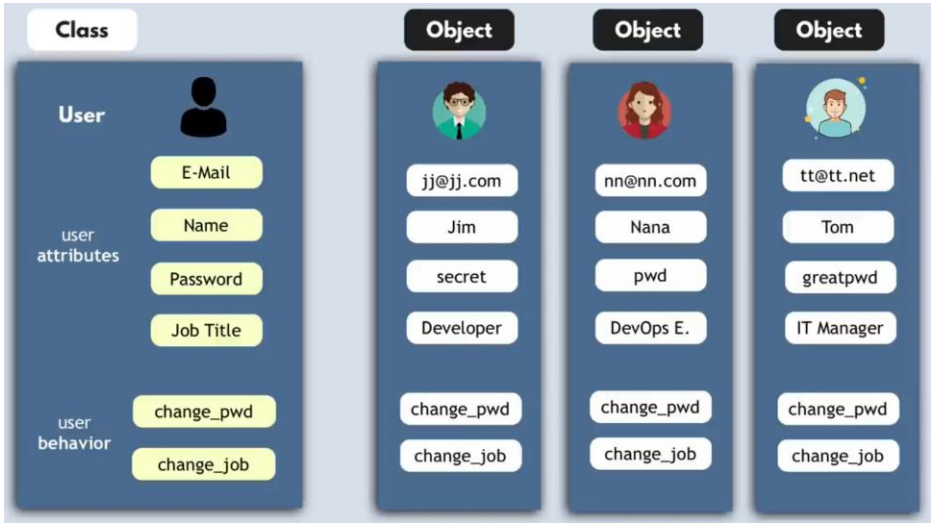
Time Stamp
Unix time stamp format
```
# get current date
now = datetime.now()
# convert current date into timestamp
timestamp = datetime.timestamp(now)

print("Date and Time :", now)
print("Timestamp:", timestamp)
```

output:
```
Date and Time : 2022-12-12 09:54:39.716420
Timestamp: 1670819079.71642
```

Timestamp → date and time object
```
timestamp = 1670819192.822178
#convert timestamp to datetime object
date_obj = datetime.fromtimestamp(timestamp)
print("Today's date & time:", date_obj)
```

output:
```
Today's date & time: 2022-12-12 09:56:32.822178
```

// Class and Objects



Several objects can be declared under a specific class
Each object has attributes specified in the class under which it is defined → A class is called as an object constructor
All classes have __init__() function → special function in python
__init__() will be executed automatically whenever an object is created under a specific class

Defining a class

```
class user:
        def __init__(self, user_email, user_name, user_jobtitle):
                self.email = user_email
                self.name = user_name
                self.current_job_title = user_jobtitle
## self refers to the class itself
## parameters adjacent to the self are values given to a specific object
## class attributes are referenced using self and a dot operator

# Defining methods
        def change_job_title(self, new_job_title)
                self. current_job_title = new_job_title
## self is used for accessing the object content

# Defining method to display user information
        def display_info(self)
                print(f"The user { self.name} is working as { self.current_job_title}. Mail id: { self.email ")}

# Defining an object
user_one = user("abc@de.com", "qwerty", "programmer")
## Order of giving values is defined in __init__() function

# Calling a method
user_one.display_info()
## Dot operator is used to call the method pertaining to the specified object
user_one. change_job_title ("Deveops")
```