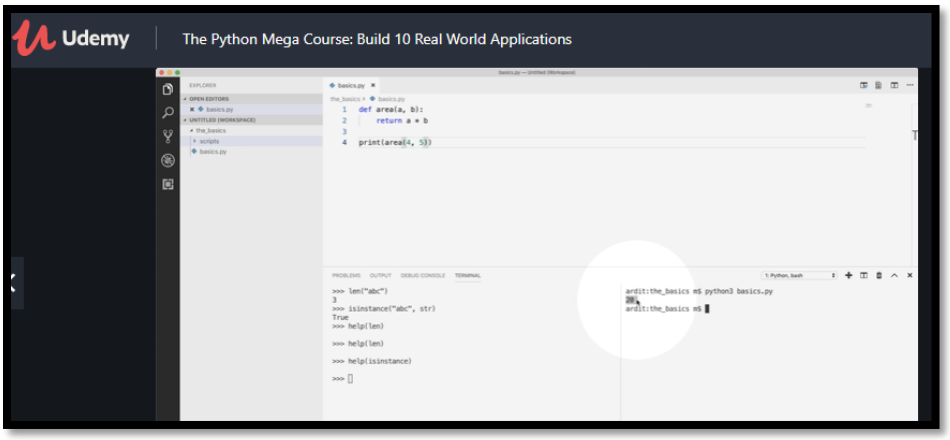
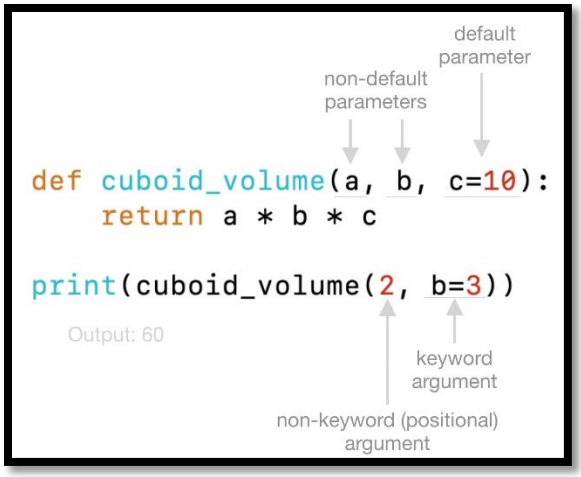
DAILY ASSESSMENT FORMAT

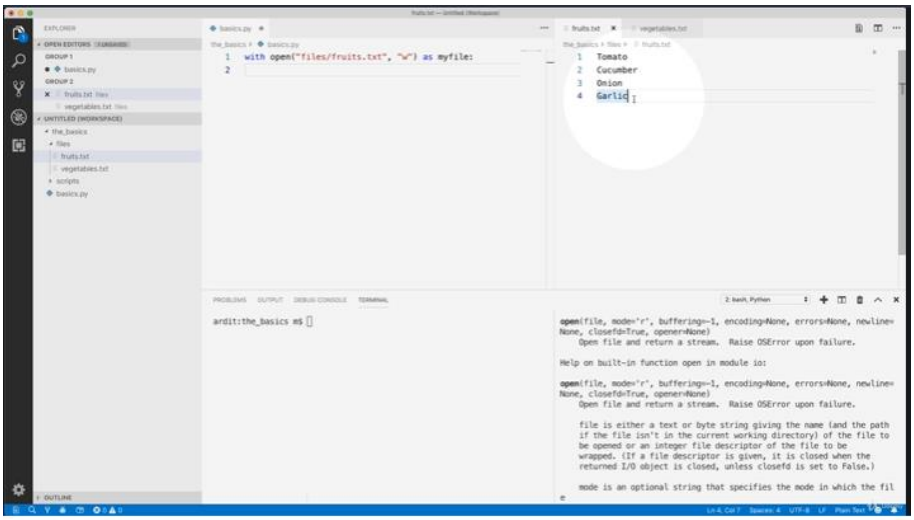
Date:	**19/05/2020**	**Name:**	**Mahima Shetty**
Course:	**Python**	**USN:**	**4AL15EC045**
Topic:	**List Comprehensions More on Functions File Processing Imported Modules**	**Semester & Section:**	**8th - A**
GitHub Repository:	**Mahima**		

Afternoon Course Details

Image of session







Report –

- A list comprehension is an expression that creates a list by iterating over another container.
- A basic list comprehension:

1. [i*2 for i in [1, 5, 10]]

   Output: [2, 10, 20]

- List comprehension with if condition:

1. [i*2 for i in [1, -2, 10] if i>0]

   Output: [2, 20]

- List comprehension with an if and else condition:

1. [i*2 if i>0 else 0 for i in [1, -2, 10]]

   Output: [2, 0, 2

If –Else condition comprehension

obj = ["Even" if i%2==0 else "Odd" for i in range(10)]

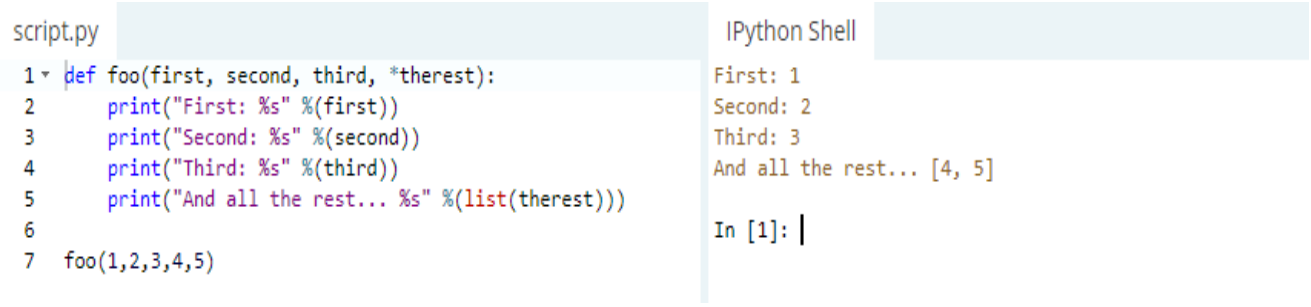
print (obj)

Output -['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']

['Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd', 'Even', 'Odd']

More on Functions

Multiple Function Arguments –



In this section I learned that:

- Functions can have more than one parameter:

```
def volume(a, b, c):
    return a * b * c
```

- Functions can have default parameters (e.g. `coefficient`):

```
def converter(feet, coefficient = 3.2808):
    meters = feet / coefficient
    return meters

print(converter(10))
```

Output: `3.0480370641306997`

Arguments can be passed as non-keyword (positional) arguments (e.g. `a`) or keyword arguments (e.g. `b=2` and `c=10`):

```
def volume(a, b, c):
    return a * b * c

print(volume(1, b=2, c=10))
```

- An *args parameter allows the function to be called with an arbitrary number of non-keyword arguments:

```
def find_max(*args):
    return max(args)
print(find_max(3, 99, 1001, 2, 8))
```

Output: `1001`

- An **kwargs parameter allows the function to be called with an arbitrary number of keyword arguments:

```
def find_winner(**kwargs):
    return max(kwargs, key = kwargs.get)
```

3.
4. print(find_winner(Andy = 17, Marry = 19, Sim = 45, Kae = 34))

   Output: `Sim`

   File Processing
   In this section I learned that:

- You can **read** an existing file with Python:

```
with open("file.txt")as file:
    content =file.read()
```

- You can **create** a new file with Python and **write** some text on it:

```
with open("file.txt","w")as file:
    content =file.write("Sample text")
```

- You can **append** text to an existing file without overwriting it:

```
with open("file.txt","a")as file:
    content =file.write("More sample text")
```

- You can both **append and read** a file with:

```
with open("file.txt","a+")as file:
    content =file.write("Even more sample text")
file.seek(0)
    content =file.read()
```

   <u>Imported Modules</u>

   In this section you learned that:

- **Built-in objects** are all objects that are written inside the Python interpreter in C language.

- **Built-in modules** contain built-ins objects.

- Some built-in objects are not immediately available in the global namespace. They are parts of a built-in module. To use those objects the module needs to be **imported** first. E.g.:

```
1. import time
2. time.sleep(5)
```

- **A list of all built-in modules** can be printed out with:

```
1. import sys
2. sys.builtin_module_names
```

- **Standard libraries** is a jargon that includes both built-in modules written in C and also modules written in Python.

- **Standard libraries** written in Python reside in the Python installation directory as *.py* files. You can find their directory path with `sys.prefix`.

- **Packages** are a collection of *.py* modules.

- **Third-party libraries** are packages or modules written by third-party persons (not the Python core development team).

- Third-party libraries can be **installed** from the terminal/command line:

  Windows:

  `pip install pandas` or use `python -m pip install pandas` if that doesn't work.

- Mac and Linux: - `pip3 install pandas` or use `python3 -m pip install pandas` if that doesn't work.