Python Basics Review 2

Multi-tasking

- **Multiprocessing**
  - Use module `multiprocessing`
  - Sample code pattern

```
def work_func(work_func_args):
    ...

l_task_ins = []
for task_id in range(task_cnt):
    task_ins = multiprocessing.Process(
                        target=work_func,
                        args=word_func_args,
                        name=task_name)
    task_ins.start()
    l_task_ins.append(task_ins)

while len(l_task_ins) > 0:
    for task_ins in l_task_ins:
        if task_ins.is_alive():
            task_ins.join(1)
        else:
            l_task_ins.remove(task_ins)
```

  - Each task runs in an individual process. Fully parallelized.
  - Processes do not share memory in between by default. Inter-Process Communication (IPC) mechanisms are required for sharing memory.
  - How many processes should be created?
- **Multithreading**
  - Use module `threading`
  - Same code pattern, except that we use `threading.Thread` to create thread instances.
  - May not be fully parallelized because of the Global Interpreter Lock (GIL). Better than single-thread though.
  - A process can have multiple threads.
  - Threads within a process do share memory.
- **Async/Await**
  - Use module `asyncio` and keywords `async` and `await`

- Single-thread multitasking.
- Change the execution order, i.e. tasks (which usually are implemented by functions) are run in a round-robin manner instead of one-after-another.

Context Managers

- **Create a context manager**
  - Sample code pattern

```
class MyContextManager:
    def __init__(self, init_args):
        ...

    def __enter__(self):
        ...
        return my_cntxt_mgr

    def __exit__(self, exc_type, exc_val, exc_tb):
        ...
```

- **Use a context manager**
  - Sample code pattern

```
# `my_cntxt_mgr` may not be the instance of MyContextManager,
#but the returned value from __enter__
with MyContextManager(...) as my_cntxt_mgr:
    ...
```

- **File IO**
  - Sample code pattern for read/write

```
with open(FILE_PATH, READ_WRITE_INDICATOR) as fd:
    # For read
    for ln_idx, ln in enumerate(fd):
        ...
    # For write
    out_str = ...
    fd.write(out_str)
```

  - If the context managers are not used, then `fd` needs to be closed manually.
    - Sample code patter

```
fd.close()
```

    - What would happen if we do not close it? (Not a part of this class)

Objects as Arguments

- Objects can be passed in functions as input arguments.
- What is passed in a function is NOT a copy of the object but the reference (i.e. there is only ONE copy of the object in the memory). And, when the object is modified, the changes are directly applied to the memory of the object (i.e. if it changes in a place, it changes everywhere).
- An object can be of any type such as `int`, `list`, `MyClass` and any others.
- `id()` can be used to check if the object is a copy or the original.

Inter-Process Communication (IPC)

- To make processes be able to communicate with each other. Processes can be on the same machine or difference machines.
- Many techniques. We focus on the networking methods.
- Use module `socket`
- Sample code patter for servers

```
# Handle each received message
def msg_handler(msg):
    ...

# Create a socket instance
serv_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
# Bind the socket instance with the network address of the server
serv_sock.bind((IP_ADDR, PORT))
# Receive messages
while True:
    msg, addr = serv_sock.recvfrom(BUFF_SIZE)
    msg_handler(msg)
```

- Sample code pattern for clients

```
client_sock = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
client_sock.sendto(MESSAGE, SERV_ADDR)
```